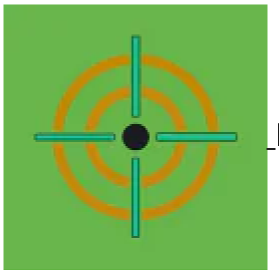
Deliver a Strike by Reversing a Badger: Brute Ratel Detection and Analysis

splunk.com/en_us/blog/security/deliver-a-strike-by-reversing-a-badger-brute-ratel-detection-and-analysis.html

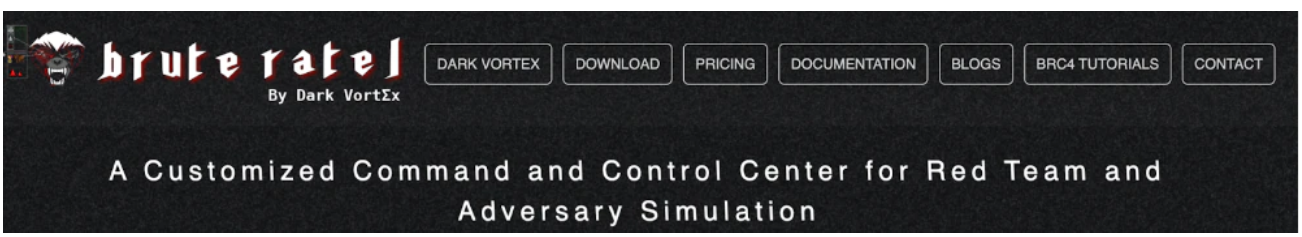
October 4, 2022

SECURITY



By Splunk Threat Research Team October 04, 2022

A new adversary simulation tool is steadily growing in the ranks of popularity among red teamers and most recently adversaries. Brute Ratel states on its website that it "is the most advanced Red Team & Adversary Simulation Software in the current C2 Market." Many of these products are marketed to assist blue teams in validating detection, prevention, and gaps of coverage. Brute Ratel goes a level further in receiving consistent updates to evade modern host-based security controls — a cat and mouse game. Adversaries pick up on these products quickly, as noted in a recent blog post by Team Cymru; Brute Ratel C4 (BRC4) servers are limited on the internet compared to other offensive security tools like Cobalt Strike and Metasploit, but its popularity is growing.



As enterprise defenders who may or may not have access to these products, we have to be able to understand the operation of the tool and its procedures and behaviors.

In this blog, the Splunk Threat Research Team (STRT) will highlight how we utilized other public research to capture Brute Ratel Badgers (agents) and create a Yara rule to help identify more on VirusTotal. Additionally, we reversed a sample to better understand its functions. STRT simulated a badger's functionality using a newly released defender-driven C2 utility. Lastly, STRT describes analytics to help defenders identify behaviors related to Brute Ratel.

Analysis

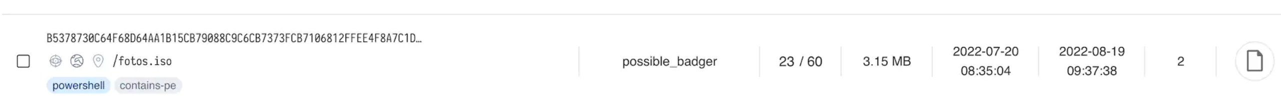
Hunting for a Badger

Brute Ratel is a commercial C2 framework available only to paying customers; yet, STRT needed a way to acquire a sample for analysis. Fortunately for us, security researchers like Spookysec.net, Unit42 and Mdsec have already found samples and blogged about their analysis. STRT leveraged the sample found on the Analyzing a Brute Ratel Badger blog post and created an experimental generic Yara rule that can be used on VirusTotal to hunt for other potential uploaded samples.

```
rule possible_badger
{
  strings:
       //mov eax, 0x00
       // push eax
       //mov eax, 0x00
       // push eax
       //mov eax, 0x00
       // push eax
       //mov eax, 0x00
       // push eax
       //mov eax, 0x00
       // push eax
       //mov eax, 0x00
       // push eax
       $code = { B8 00 00 00 00 50 B8 00 00 00 00 50 B8 00 00 00 00 50 B8 00 00 00
00 50 B8 00 00 00 00 50 B8 00 00 00 00 50}
  condition:
      all of them
}
```

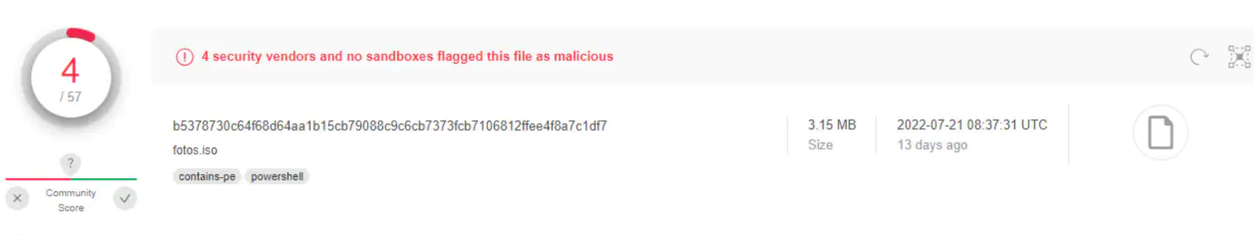
The Yara rule above hunts for a series of move zero bytes instructions to the EAX register which are then pushed to the stack. These instructions were identified as part of the initial shellcode that sets up the BRC4 agent DLL module on the stack.

The figure below shows one of the files flagged by the Yara rule, an ISO file named fotos.iso.

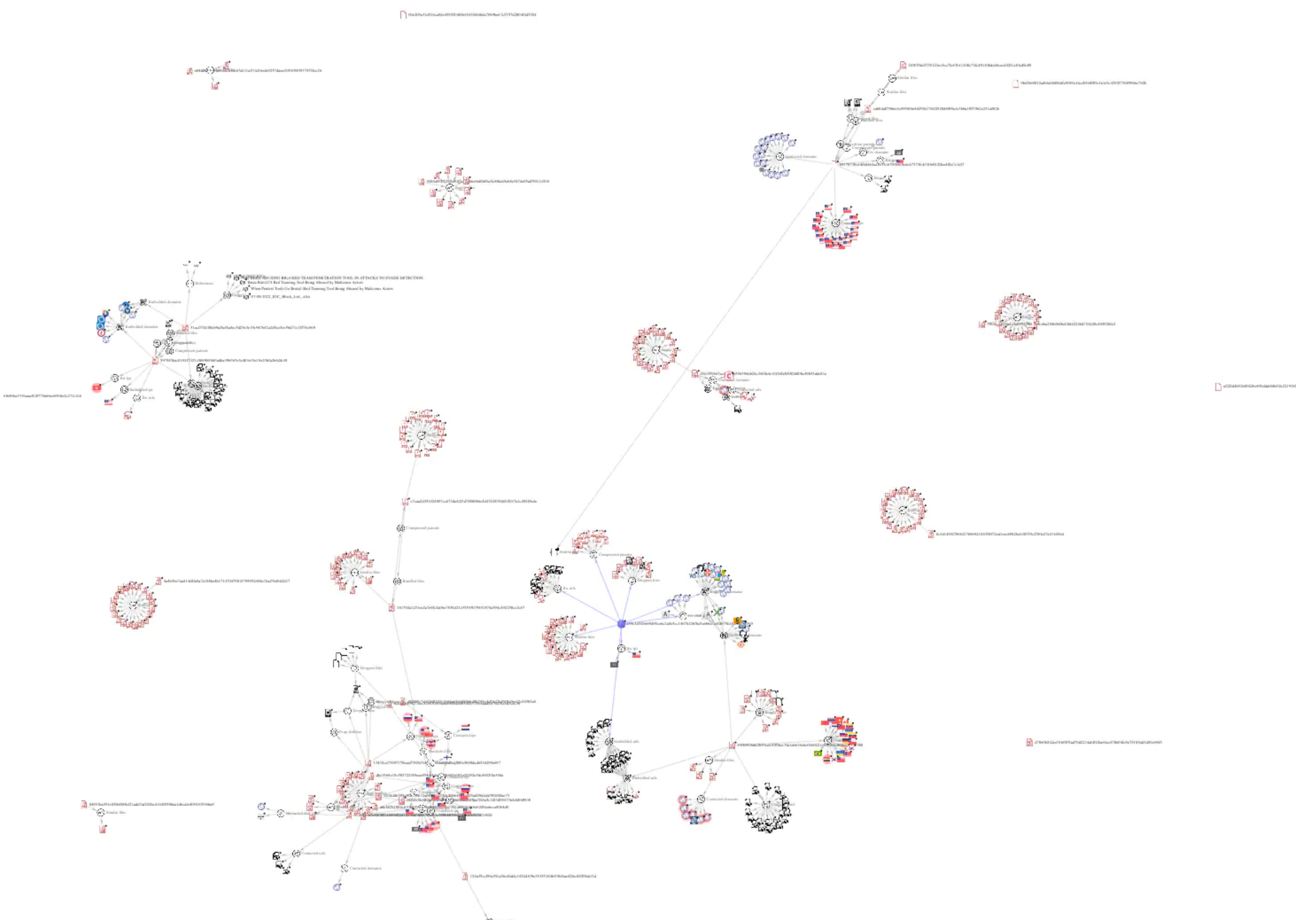


The first submission of this file in VT was on July 20, 2022, from Poland.

The figure below shows the VirusTotal detection list of the ISO fotos.iso at the time of writing.



Using the Yara rule we were able to identify 31 similar samples and graph them using VirusTotal Graphs.



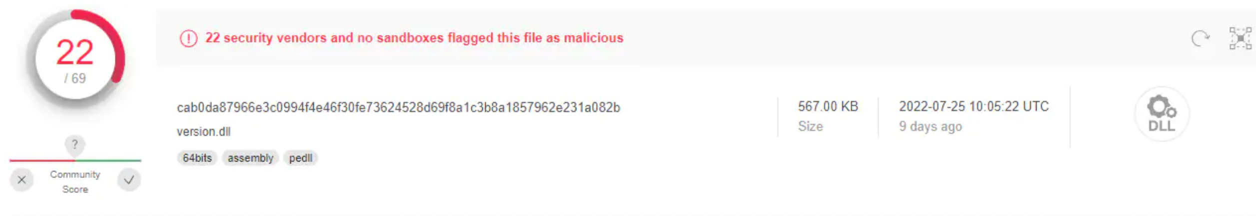
The full VirusTotal graph may be found here.

Malicious ISO File

ISO containers are a common way to deliver malware among threat actors. It enables them to archive malicious files and even bypass security features such as the Mark-of-the-Web. The found sample contains the legitimate Microsoft signed OneDrive binary renamed as onedrive_fotos.exe as well as two hidden DLLs: version.dll and versions.dll files. The latter is also a Microsoft-signed legitimate DLL while the first one is a malicious library that will execute the BRC4 agent.

This initial access vector leverages the DLL Side-Loading technique (T1574.002) to obtain code execution on the victim host. Side-loading takes advantage of the DLL search order used by the loader by positioning both the victim application and malicious payload alongside each other. When the victim mounts the ISO and executes the onedrive_fotos.exe binary, it will load the maliciously crafted version.dll.

The figure below shows the VirusTotal detection list of the version.dll library at the time of writing.



The ISO file we analyzed is similar to the sample analyzed by Palo Alto's Unit42 in their blog post covering Brute Ratel with a few notable differences:

- This ISO does not contain a shortcut LNK file and relies on the victim double clicking the onedrive_fotos.exe binary to load the malicious DLL.
- The initial shellcode is embedded in the hidden DLL and not present as another file in the ISO archive.

The following image provides a high level overview of the initial access attack vector.

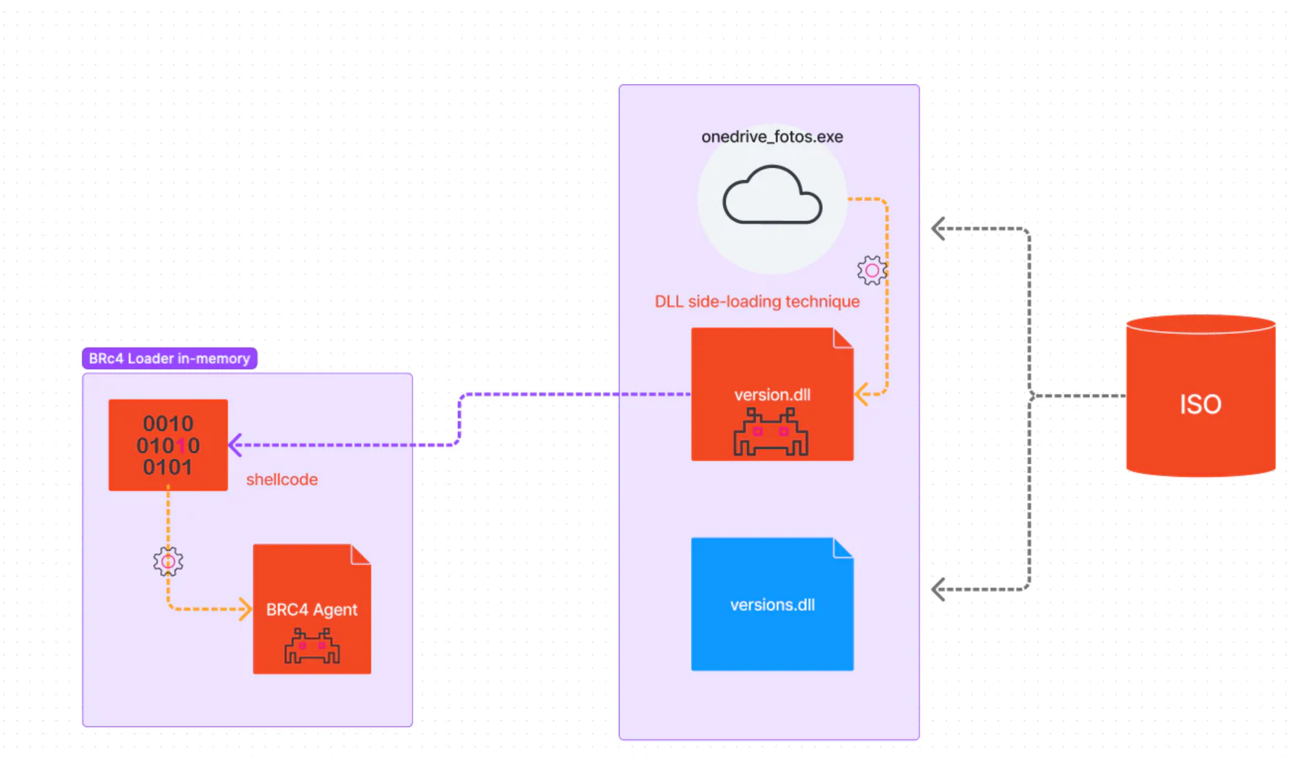
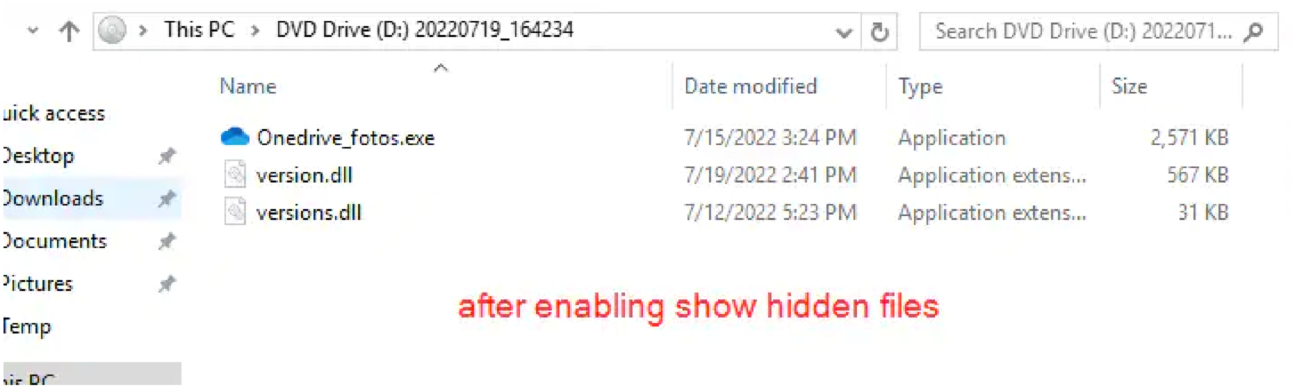


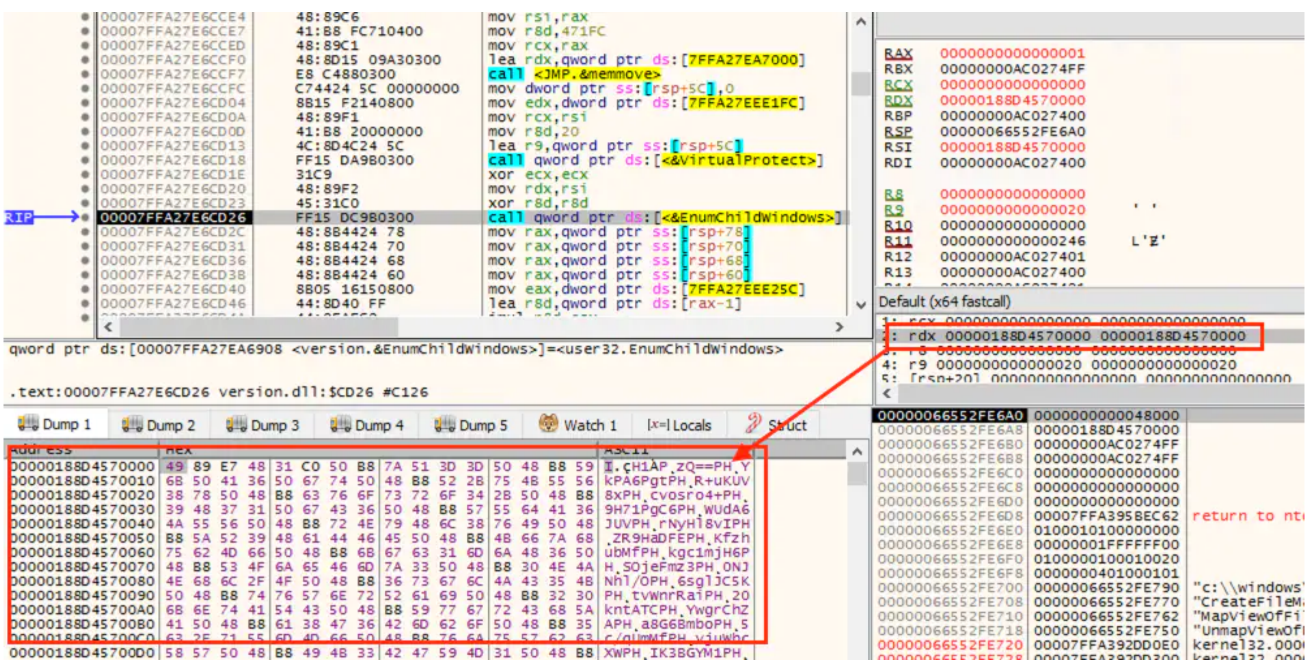
Figure 2.1 and Figure 2.2 show the .ISO component files before and after enabling the “Show Hidden Files” setting.



Initial Shellcode Execution

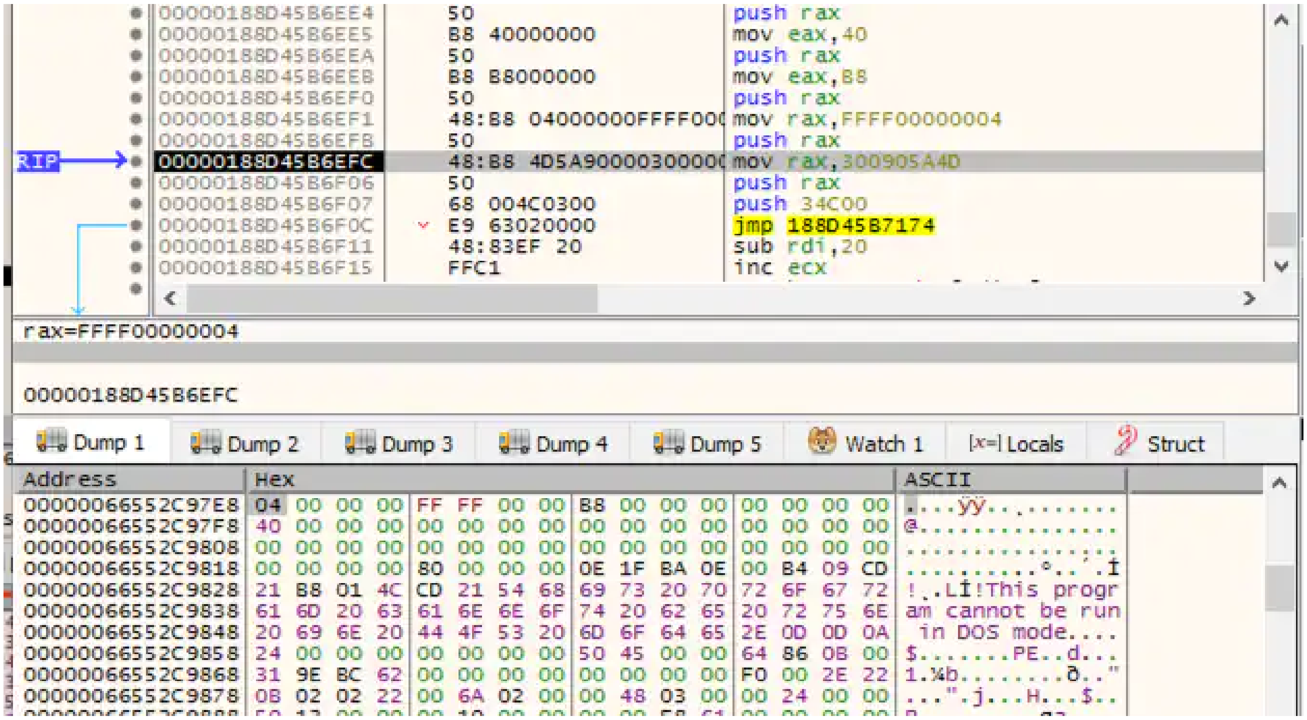
The malicious version.dll file has an embedded unencrypted shellcode in its .data section that will be copied to an allocated memory address space with the PAGE_EXECUTE_READ protection to then be executed using the callback function of the EnumChildWindows Windows API. This shellcode execution technique was first seen being used by the Lazarus group.

Figure 3 shows the code snippet of the EnumChildWindows callback function used to execute the shellcode.



The shellcode will set up the Brute Ratel C4 DLL agent in the memory stack using several push mnemonics. Afterward, the shellcode will allocate an executable memory page space where it will move the DLL agent from the stack byte per byte. Lastly, It will execute it using the undocumented API NtCreateThreadEx.

Figure 4 is the code showing the last push command executed by the shellcode to finalize copying the BRC4 DLL agent to the stack.



Once the DLL agent is placed in the executable memory page, we can export it to disk to perform static analysis. We used the Detect It Easy tool to perform high level analysis of the extracted BRC4 DLL and obtain information such as the exported functions (Figure 4.1), the entropy of the file and each section (Figure 4.2), etc.

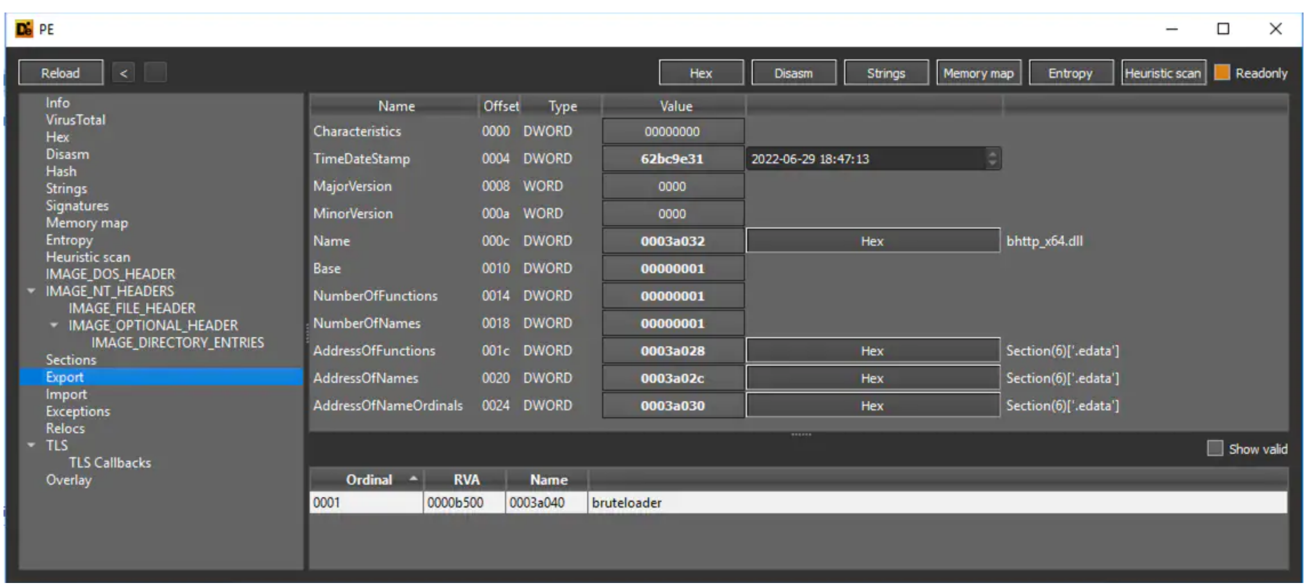


Figure 4.1

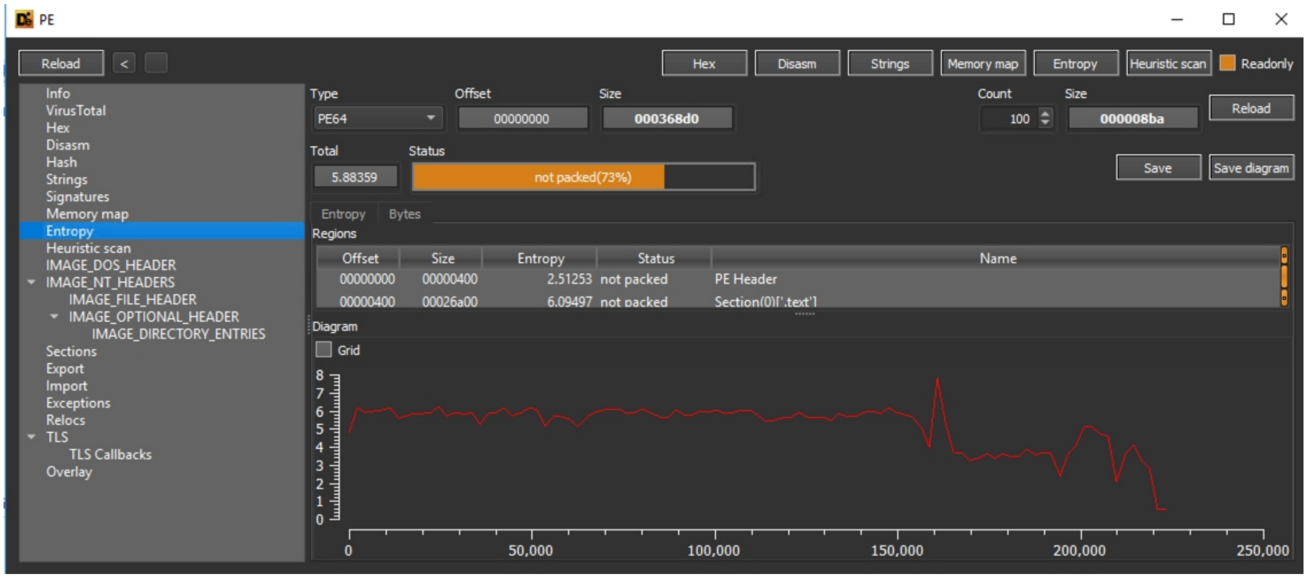


Figure 4.2

Figure 5 shows the code that runs a syscall function to execute the NtCreateThreadEx Windows API with the startAddress argument pointing to the “bruteloader” export function of the BRC4 DLL agent loaded in memory. This thread also has an argument that points to the encoded and encrypted initial configuration that will be used for its C2 communication and beaconing.

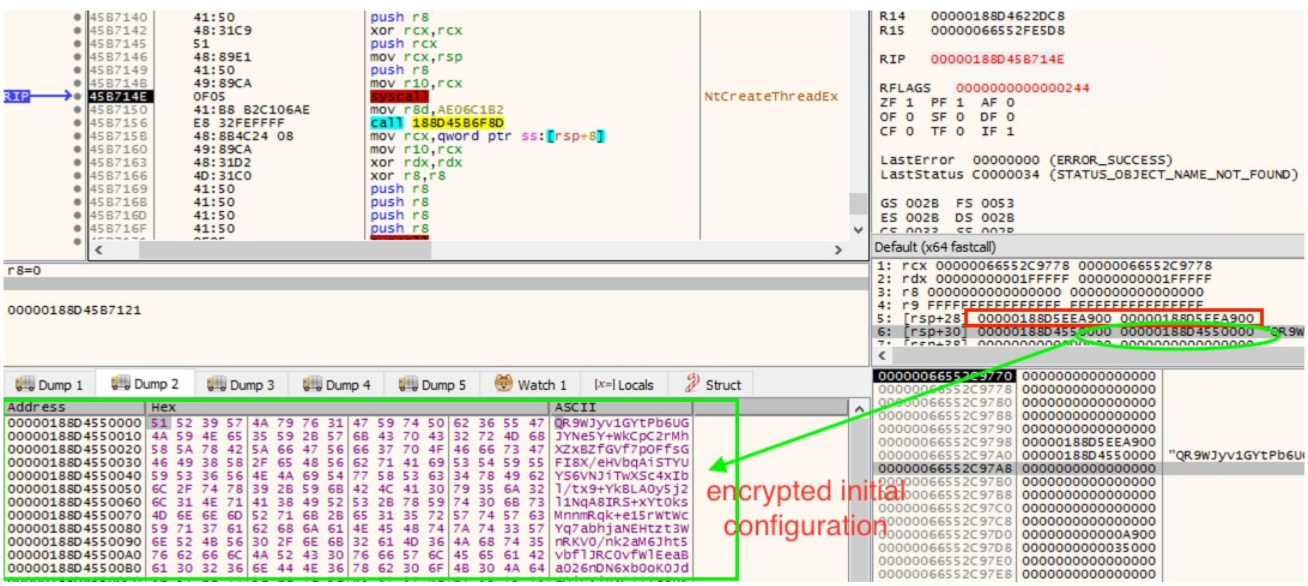
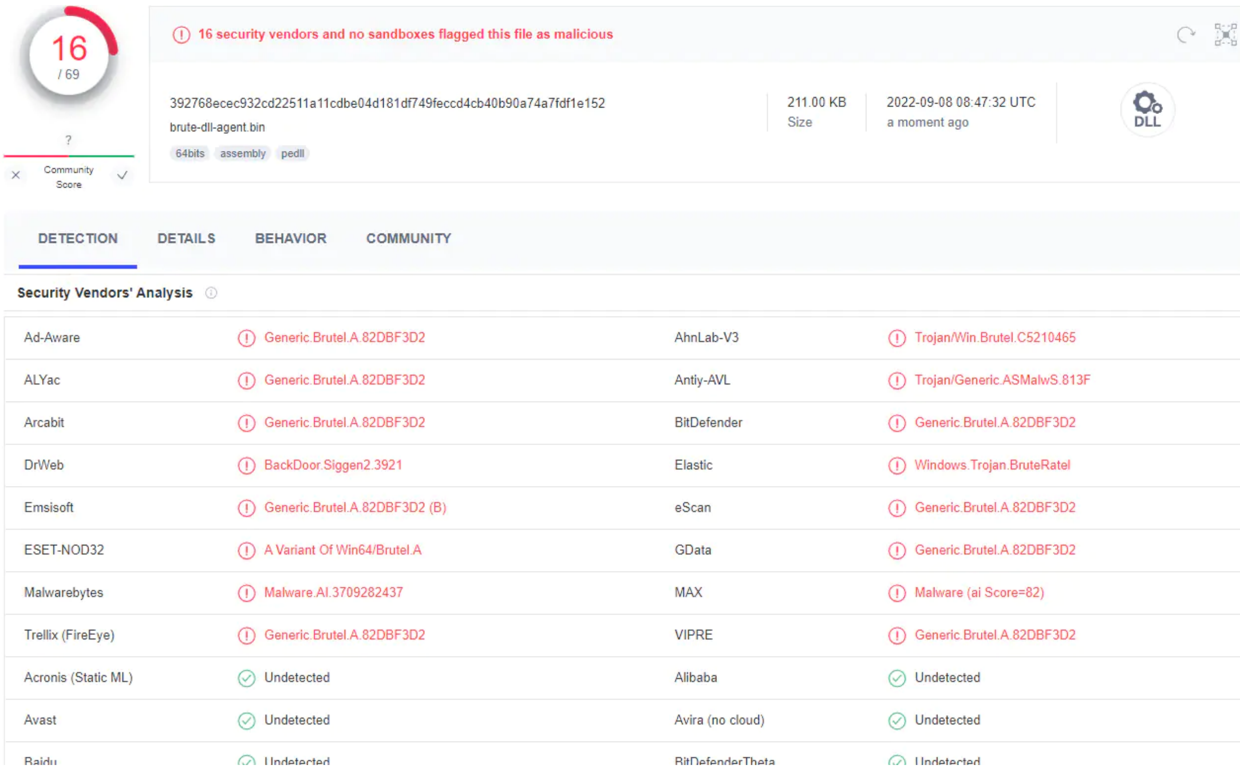


Figure 5

The DLL module we extracted from memory was submitted to VT in this link and can be seen in Figure 5.1.



BRC4 DLL Agent Module

Initial Configuration

The configuration data is encoded with base64 and encrypted with RC4 with the passphrase key “bYXJm/3#M?:XyMBF”. Figure 6 is the decrypted version of this configuration data that contains the command and control servers, port (HTTPS), user agent, cookie, and many more details.

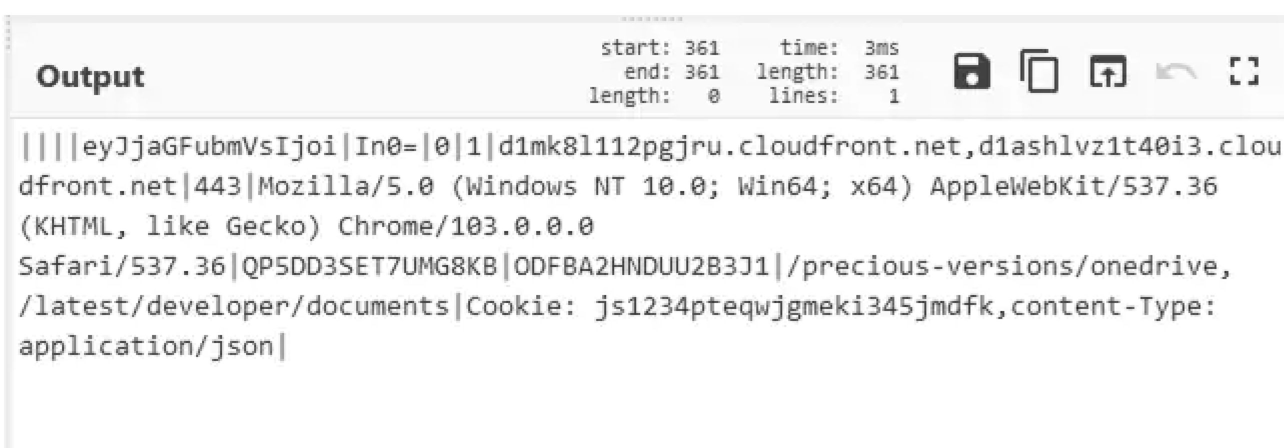


Figure 6

The Brute Ratel DLL agent used by this malicious version.dll is composed of techniques to evade detection from endpoint detection and response (EDR), antivirus products, and even obfuscation and encryption to thwart static code analysis.

The following section describes some of the capabilities the STRT found during our analysis of the BRC4 DLL module embedded in version.dll which include: gaining elevated privileges, collecting sensitive information, evading detections, and dumping processes, among others.

BRc4 Agent Capabilities

Windows API Abuse

The BRC4 agent employs several techniques to invoke and abuse native Windows APIs. To attempt to bypass security solutions that rely on hooking common APIs, BRCc4 makes use of direct system calls. The BRC4 agent can also dynamically resolve functions memory addresses using pre-computed hardcoded hashes. Figure 6.1 shows the code snippet of its hashing algorithm it uses in parsing its needed API upon traversing the export table of its needed DLL modules.

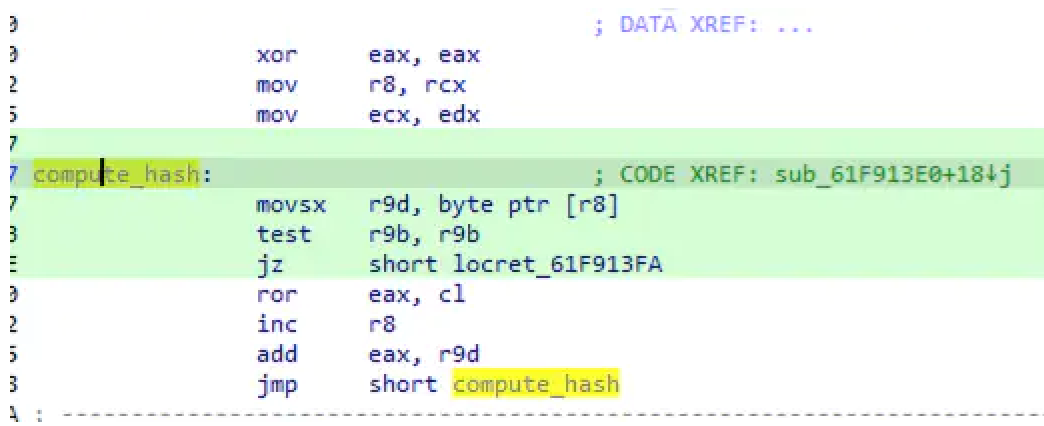


Figure 6.1

Common functionality implemented by C2 implants is the ability to verify the connectivity with another host using the ICMP protocol. The BRC4 DLL agent implements this by using the native Windows API IcmpCreateFile and IcmpSendEcho.

Figure 7 shows the code with the resolved API hash value on how to implement a PING to a target host using those Windows APIs.

```
IcmpHandle = dw_var_IcmpCreateFile_HASH();
if ( !IcmpHandle )
  return 0i64;
if ( dw_var_inet_pton_HASH(AF_INET, pszIPAddrs, &pAddrBuf) == 1 )
{
  ReplyBuffer = calloc(0x31ui64, 1ui64);
  if ( dw_var_IcmpSendEcho_HASH(IcmpHandle, pAddrBuf, &RequestData, 1i64, 0i64, ReplyBuffer, 0x31, 10000)
    && ReplyBuffer->DataSize )
  {
    possible_wsprintf(Block, "[", pszIPAddrs, ReplyBuffer->Options.Ttl);
  }
  else
  {
    possible_wsprintf(Block, L"[-] Unreachable: %S\n", pszIPAddrs);
  }
}
else
{
  ReplyBuffer = 0i64;
  possible_wsprintf(Block, L"[-] Bad IP\n");
```

Figure 7

SeDebugPrivilege

By default, users can debug only the processes that they own. In order to debug other processes or processes owned by other users, a process needs to have a SeDebugPrivilege privilege token. This privilege token is abused by adversaries to elevate process access to inject malicious code or dump processes. Figure 8 shows how BRC4 adjusts the token privilege of its process to gain debug privileges.

```
mov     qword ptr [rsp+68h+SeDebugPrivilege_str+8], rax
call    cs:dw_var_OpenProcessToken_HASH
test    eax, eax
jz      short loc_61F92306
xor     ecx, ecx        ; _QWORD
lea     rbx, [rsp+68h+tokenpriv]
lea     rdx, [rsp+68h+SeDebugPrivilege_str] ; _QWORD
lea     r8, [rsp+68h+tokenpriv.Privileges] ; _QWORD
call    cs:dw_var_LookupPrivilegeValueA_HASH
test    eax, eax
jz      short loc_61F922F9
xor     edx, edx        ; _QWORD
mov     rcx, [rsp+68h+hToken] ; _QWORD
mov     r9d, 10h        ; _QWORD
mov     r8, rbx         ; _QWORD
mov     [rsp+68h+tokenpriv.PrivilegeCount], 1
mov     [rsp+68h+tokenpriv.Privileges.Attributes], 2
mov     [rsp+68h+var_40], 0 ; _QWORD
mov     [rsp+68h+var_48], 0 ; _QWORD
call    cs:dw_var_AdjustTokenPrivileges_HASH
test    eax, eax
jz      short loc_61F922F9
mov     rcx, [rsp+68h+hToken] ; _QWORD
call    cs:dw_var_CloseHandle_HASH
```

Figure 8

Parse Clipboard Data

Figure 9 shows the code snippet BRC4 uses to parse or copy the clipboard data on the targeted host using the Windows API functions OpenClipboard and GetClipboardData.

```
sub     rsp, 40h
mov     [rsp+58h+Block], 0
mov     r13, rcx
xor     ecx, ecx
lea     r14, [rsp+58h+Block]
call    cs:dw_var_OpenClipboard_HASH
test    eax, eax
jz      short loc_61F95B3E
mov     ecx, 0Dh
call    cs:dw_var_GetClipboardData_HASH
mov     r12, rax
test    rax, rax
jz      short loc_61F95B38
mov     rcx, rax
call    cs:dw_var_GlobalLock_HASH
test    rax, rax
jnz     short loc_61F95B58

loc_61F95B38:                           ; CODE XREF: mw_get_clipboard+38↑j
        call    cs:dw_var_CloseClipboard_HASH

loc_61F95B3E:                           ; CODE XREF: mw_get_clipboard+25↑j
        call    cs:dw_var_GetLastError_HASH
```

Figure 9

Retrieve DNS CACHE RECORD

Figure 10 shows the code snippet implemented by BRC4 to parse the DNS cache record of the infected host using the undocumented DnsGetcacheDataTable Windows API.

```
push    r14
push    r13
push    r12
push    rsi
push    rbx
sub     rsp, 30h
mov     edx, 18h        ; Size
mov     r14, rcx
mov     ecx, 1          ; Count
lea     r13, [rsp+58h+Block]
mov     [rsp+58h+Block], 0
call    calloc
mov     r12, rax
mov     rcx, rax
call    cs:dw_var_DnsGetCacheDataTable_HASH
mov     rbx, [r12]
test    rbx, rbx
jz      short loc_61F94D60
lea     rdx, asc_61FACF7E ; "["
mov     rcx, r13
lea     rsi, asc_61FAC3B2 ; " "
call    possible_wsprintf
```

Figure 10

Duplicate Token

Token manipulation is a technique used to create a new process with a token “taken” or “duplicated” from another process. This is a common technique leveraged by adversaries, red teamers, and malware families to elevate the privileges of their processes.

Figure 11 shows the code function that duplicates the token of “winlogon.exe” or “logonui.exe” and uses that token to a new process using CreateProcessWithTokenW API.

```
  --v10;
}
if ( dw_var_OpenProcessToken_HASH(v18, 10i64, &v19) )
{
  if ( dw_var_DuplicateTokenEx_HASH(v19, 395i64, 0i64, 2i64, 1, &v20) )
  {
    v12 = 130i64;
    v13 = v33;
    while ( v12 )
    {
      *v13 = 0;
      v13 += 2;
      --v12;
    }
    sub_61F91B90(v33, 260i64, asc_61FAD76A, a2);
    if ( dw_var_CreateProcessWithTokenW_HASH(v20, 1i64, 0i64, v33, 0x8000000, 0i64, 0i64, v31, v23) )
    {
      possible_wsprintf(&Block, "[", v24);
      dw_var_CloseHandle_HASH(v23[1]);
      dw_var_CloseHandle_HASH(v23[0]);
      goto LABEL_34;
```

Figure 11

Patch ETWEventWrite API

Another interesting feature of the BRC4 DLL agent is that it can evade Event Tracing for Windows (ETW) and AMSI Windows mechanisms by patching known API responsible for generating or tracing system events.

Figure 12 shows the code of this DLL agent that patches “EtwEventWrite” API with “0xC3” opcode which is a “return instruction” to evade the ETW event trace logging.

```
                push    r12
                sub     rsp, 30h
                mov     ecx, 3CFA685Dh
                mov     [rsp+38h+var_C], 0C3h ; 'Ã'
                call    mw_resolve_dll_name
                mov     ecx, 2047C3EEh
                mov     rdx, rax
                call    mw_resolve_api_name
                mov     r12, rax
                test    rax, rax
                jnz     short loc_61F94A51

loc_61F94A4D:                           ; CODE XREF: mw_EtwEventWrite_ret_
                xor     eax, eax
                jmp     short loc_61F94AA1
; ---------------------------------------------------------------------------

loc_61F94A51:                           ; CODE XREF: mw_EtwEventWrite_ret_
                lea     r9, [rsp+38h+lpflOldProtect]
                mov     r8d, 4
                mov     edx, 4
                mov     rcx, rax
                call    cs:dw_var_VirtualProtect_HASH
                test    eax, eax
                jz      short loc_61F94A4D
                lea     rdx, [rsp+38h+var_C]
                mov     r8d, 4
                mov     rcx, r12
                call    sub_61F9B6F0
                mov     r8d, [rsp+38h+lpflOldProtect]
                lea     r9, [rsp+38h+var_10]
                mov     rcx, r12
                mov     edx, 4
                call    cs:dw_var_VirtualProtect_HASH
                test    eax, eax
                setnz   al
                movzx   eax, al
```

Figure 12

Parent Process ID Spoofing

BRC4 is also capable of spoofing the parent process (PPID) for its newly created process to evade detections that are based on parent/child process relationships.

The code below in Figure 13 is the function that initializes the process attributes and thread creation for the parent process spoofing technique.

```
  dw_var_InitializeProcThreadAttributeList_HASH(0i64, 1i64, 0i64);
  v32 = v40;
  v16 = GetProcessHeap();
  v17 = HeapAlloc(v16, 8u, v32);
  v13 = v17;
  if ( !v17
    || !dw_var_InitializeProcThreadAttributeList_HASH(v17, 1i64, 0i64)
    || !dw_var_UpdateProcThreadAttribute_HASH(v13, 0i64, 131079i64, &v41, 8i64, 0i64, 0i64) )
  {
    goto LABEL_33;
  }
  v52 = v13;
  v18 = 134742016;
}
else
{
  v13 = 0i64;
  v18 = 0x8000000;
}
if ( v5 )
{
  v18 |= 4u;
  goto LABEL_23;
}
v19 = *(v1 + 3520);
if ( !v19 )
{
.ABEL_23:
  if ( (dw_var_CreateProcessA_HASH)(0i64, v3, 0i64, 0i64, 1, v18, 0i64, 0i64, v48, &v42) )
```

Figure 13

Retrieves IPV4 to Physical Address Mapping Table

Figure 14 shows the code snippet of the function that enumerates Address Resolution Protocol (ARP) entries or physical address map table for IPV4 on the local system using the GetIpNetTable Windows API.

```
Count = 0;
Count_4 = 0i64;
dw_var_GetIpNetTable_HASH(0i64, &Count, 1i64);
IpNetTable = calloc(Count, 0x1Cui64);
arp_entries = IpNetTable;
if ( !IpNetTable )
  goto LABEL_21;
retVal = dw_var_GetIpNetTable_HASH(IpNetTable, &Count, 1i64);
if ( !retVal || retVal == ERROR_NO_DATA )
{
  arp_table = arp_entries->table;
  v5 = 0;
  for ( i = 0; ; ++i )
  {
    if ( arp_entries->dwNumEntries <= i )
      goto LABEL_21;
    v7 = arp_table->dwIndex;
    if ( arp_table->dwIndex != v5 )
    {
      sub_61FA7720(&Count_4, "\n", v7);
      sub_61FA7720(&Count_4, "[", L"Internet Address", "P", "T");
    }
    dwAddr = arp_table->dwAddr;
    v21 = 0;
    Buffer = 0ui64;
    LODWORD(v15) = HIBYTE(dwAddr);
    LODWORD(v14) = BYTE2(dwAddr);
    sprintf(&Buffer, "%d.%d.%d.%d", dwAddr, BYTE1(dwAddr), v14, v15);
    sub_61FA7720(&Count_4, L"  - %-24S", &Buffer);
    v9 = arp_table->dwPhysAddrLen;
    if ( !v9 )
      break;
    *v22 = 0i64;
    v23 = 0i64;
    v24 = 0i64;
    if ( v9 == 6 )
    {
      LODWORD(v17) = arp_table->bPhysAddr[5];
      LODWORD(v16) = arp_table->bPhysAddr[4];
      LODWORD(v15) = arp_table->bPhysAddr[3];
      LODWORD(v14) = arp_table->bPhysAddr[2];
      sprintf(
        v22,
        "%02X-%02X-%02X-%02X-%02X-%02X",
        arp_table->bPhysAddr[0],
        arp_table->bPhysAddr[1],
        v14,
        v15,
        v16,
        v17);
      v10 = v22;
```

Figure 14

Below is the list of other capabilities we found in the BRC4 DLL module loaded by this malicious version.dll file:

- Check the active and idle session of the user in the target host
- TCP bind connection

- Create, copy, move and delete File
- Create, delete, move directory
- Get and set current working directory
- Get domain information
- Enumerate Drivers with their file information
- Create, start, modify, enumerate and delete services
- Get environment variable list
- Change workstation wallpaper
- Get host by name
- Enumerate logical drives
- Get process information
- Get process token privileges
- Retrieve global information for all users and groups in security databases like SAM
- List files in a directory
- Workstation lock screen
- Process minidump
- Retrieve NET BIOS information
- Process Injection (QAPC, CreateRemoteThread, and CreateSection Techniques)
- Enumerate Registries
- Get system information
- Terminate a process
- Taking windows desktop screenshot
- Execute shell command ("RUNAS")
- Retrieves the time of the last input event
- List installed software applications in the targeted host
- Retrieves the active processes on a specified RDP session

Brute Ratel Simulation

Detections written by the Splunk Threat Research Team need to pass the automated detection testing pipeline before they can be released. Building detections for some of the interesting TTPs we identified by analyzing BRC4 was no different; we needed a way to simulate these techniques in a lab environment in order to generate the datasets used for testing and stored in the Attack Data Github repository.

This presented two key challenges:

1. The C2 server of the BRC4 agent we analyzed was inaccessible or already offline during our analysis. Furthermore, even if it was online, we would have not been able to instruct the agent to execute the specific tasks we wanted to run.
2. The Brute Ratel server-side application is a commercial product and the creator was unavailable for us to write detections against the product.

Introducing Atomic-C2

To approach these challenges, we decided to write our own minimal Command & Control framework using C++ for the implant and Python for the server: Atomic-C2. This tool will never be meant to be used as part of offensive engagements but rather to be used by blue teams to learn about how C2s work and be able simulate techniques when commercial or criminal toolsets are not available.

For this initial and Proof-Of-Concept version of Atomic-C2, we took some of the techniques we learned by reverse engineering the BRC4 agent and re-wrote (simulate) them in C++ with a server side component to trigger them. Two examples are shown below.

Figure 14.1 shows how we simulate the previously shown capability to harvest or parse the clipboard data.

Figure 14.2 shows how we simulate the capability responsible for parent process ID spoofing.

As another example, Figure 15 shows the screenshot of how we simulate the technique of locking the screen of the targeted workstation. The C2 server operator runs the “lock” command to instruct the agent running in the victim host to execute the simulated workstation lock screen code.

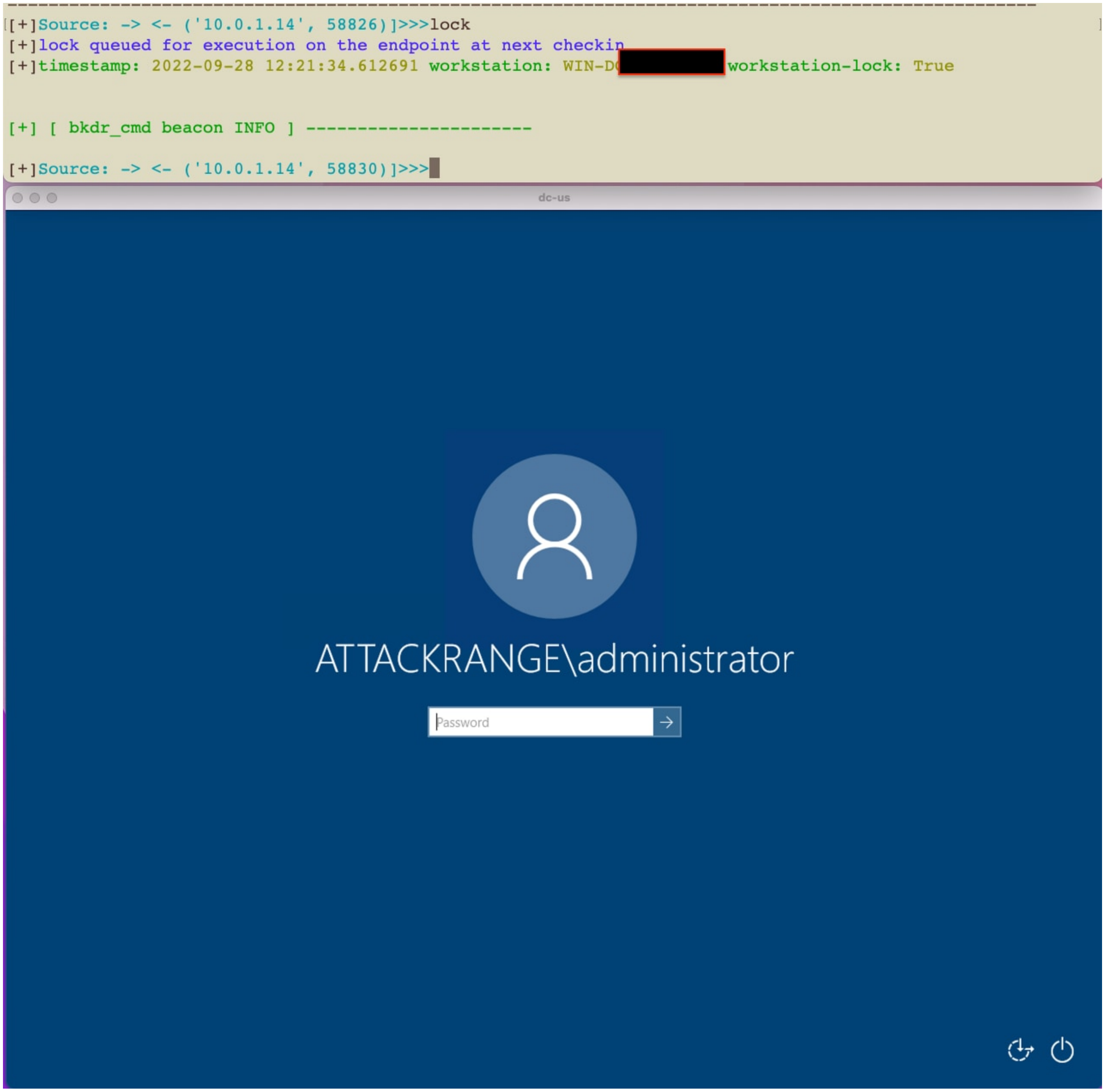
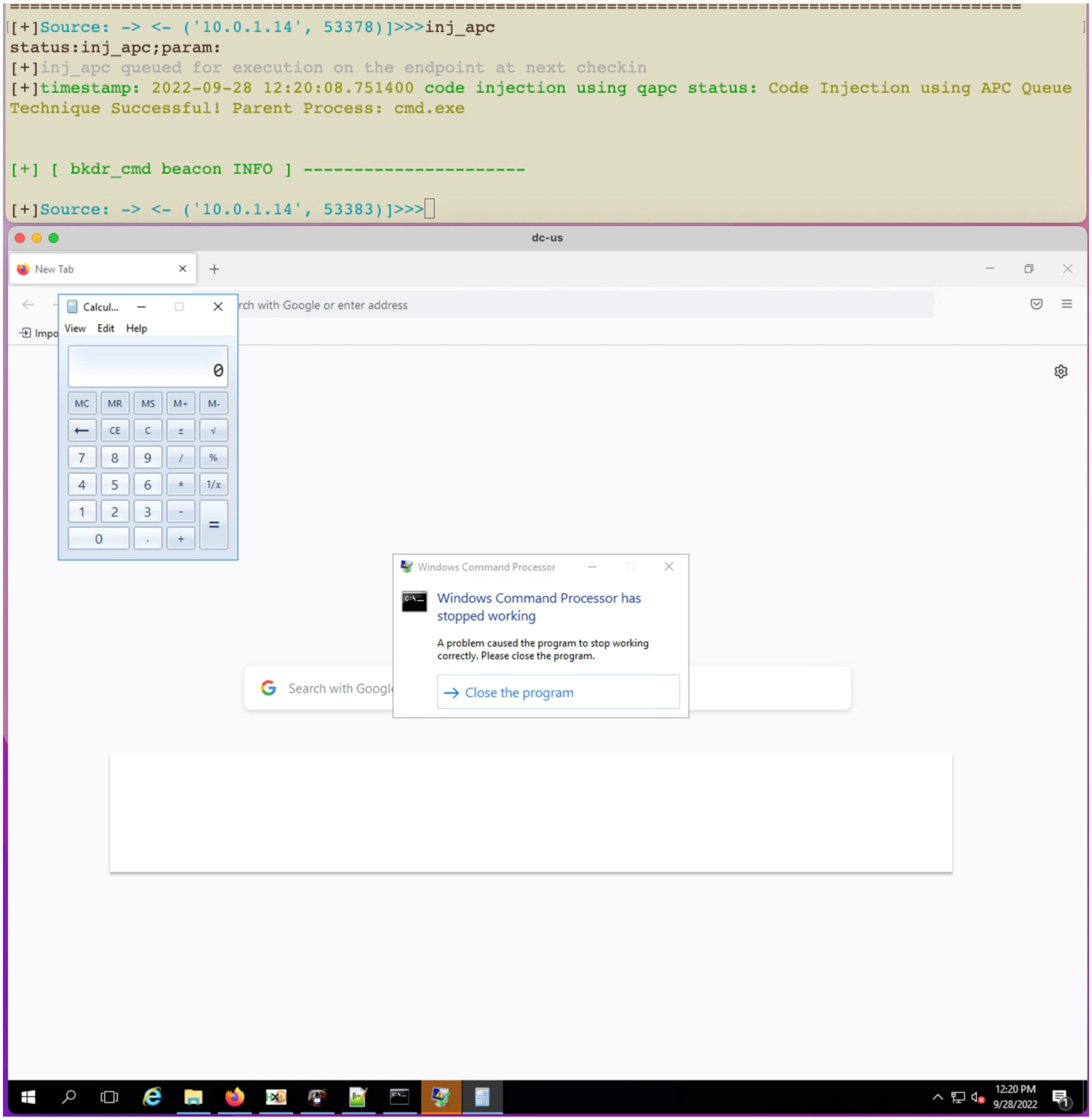


Figure 16 shows a screenshot of the simulated QUEUE APC process code injection technique. The simulated code will look for a cmd.exe process and inject shellcode that will execute a calc.exe.



Atomic-C2 helped us simulate Brute Ratel techniques to obtain the datasets we needed to create detections and to pass the automated testing process. At the moment, Atomic-C2 is an internal project, but we hope to release it in the upcoming months.

Brute Ratel C4 Analytic Story

Armed with the knowledge gained from reversing the Brute Ratel sample and the datasets generated with Atomic-C2, the Splunk Threat Research Team developed a new analytic story to help security operations center (SOC) analysts detect adversaries leveraging the Brute Ratel Command & Control framework. Specifically, the new Analytic Story introduces 17 detection analytics across 10 MITRE ATT&CK techniques.

There can be multiple approaches that rely on different data sources to hunt for Brute Ratel behavior. For this release, we wanted to focus on what we consider to be the most relevant data source: endpoint telemetry. Thus, we focused on the following data sources:

- Process Execution & Command Line Logging
- Windows Security Event Id 4688, Sysmon, or any Common Information Model compliant EDR technology.
- Windows Security Event Log
- Windows System Event Log

The next table describes the data models and the Splunk Technical Add-Ons we used to develop the detection analytics.

Sourcetype	CIM Datamodel	Technical Add-On
Sysmon	Endpoint	Splunk Add-on for Sysmon
Windows Security Events	Endpoint	Splunk Add-on for Microsoft Windows
Windows System Events	Endpoint	Splunk Add-on for Microsoft Windows

IOCs:

Name: fotos.iso Size: 3299328 bytes (3222 KiB) SHA256: b5378730c64f68d64aa1b15cb79088c9c6cb7373fcb7106812ffee4f8a7c1df7
Name: version.dll Size: 580608 bytes (567 KiB) SHA256: cab0da87966e3c0994f4e46f30fe73624528d69f8a1c3b8a1857962e231a082b
File: brute-dll-agent.bin (in-memory) Size: 216064 bytes (211.00 KB) Sha256: 392768ecec932cd22511a11cdbe04d181df749feccd4cb40b90a74a7fdf1e152
File: versions.dll Size: 31496 bytes (30.76 KB) Sha256: e549d528fee40208df2dd911c2d96b29d02df7bef9b30c93285f4a2f3e1ad5b0

File: ONEDRIVE.EXE
Size: 2632088 bytes (2.51 MB)
Sha256: a8f50e28989e21695d76f0b9ac23e14e1f8ae875ed42d98eaa427b14a7f87cd6

Automate with SOAR Playbooks

All of the previously listed detections create entries in the risk index by default, and can be used seamlessly with risk notables and the Risk Notable Playbook Pack. The community Splunk SOAR playbooks below can be used in conjunction with some of the previously described analytics:

Playbook	Description
Delete Detected Files	This playbook acts upon events where a file has been determined to be malicious (ie webshells being dropped on an end host). Before deleting the file, we run a "more" command on the file in question to extract its contents. We then run a delete on the file in question.
Internal Host WinRM Investigate	This playbook performs a general investigation on key aspects of a windows device using windows remote management. Important files related to the endpoint are generated, bundled into a zip, and copied to the container vault.
Block Indicators	This playbook retrieves IP addresses, domains, and file hashes, blocks them on various services, and adds them to specific blocklists as custom lists

Why Should You Care?

With this article we enabled security analysts, blue teamers and splunk customers to identify the TTP's used by threat actors abusing BRC4 DLL agents.

By understanding its behaviors, we were able to generate telemetry and datasets to develop and test splunk detections designed to defend and respond against this type of threats.

Learn More

You can find the latest content about security analytic stories on GitHub and in Splunkbase. Splunk Security Essentials also has all these detections available via push update.

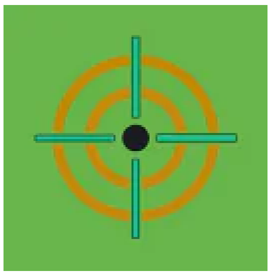
For a full list of security content, check out the release notes on Splunk Docs.

Feedback

Any feedback or requests? Feel free to put in an issue on GitHub and we'll follow up. Alternatively, join us on the Slack channel #security-research. Follow these instructions if you need an invitation to our Splunk user groups on Slack.

Contributors

We would like to thank the following for their contributions to this post:



Posted by

Splunk Threat Research Team

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the Attack Data repository.

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more Splunk Security Content.

TAGS
Cybersecurity
Show All Tags

Show Less Tags

Related Posts