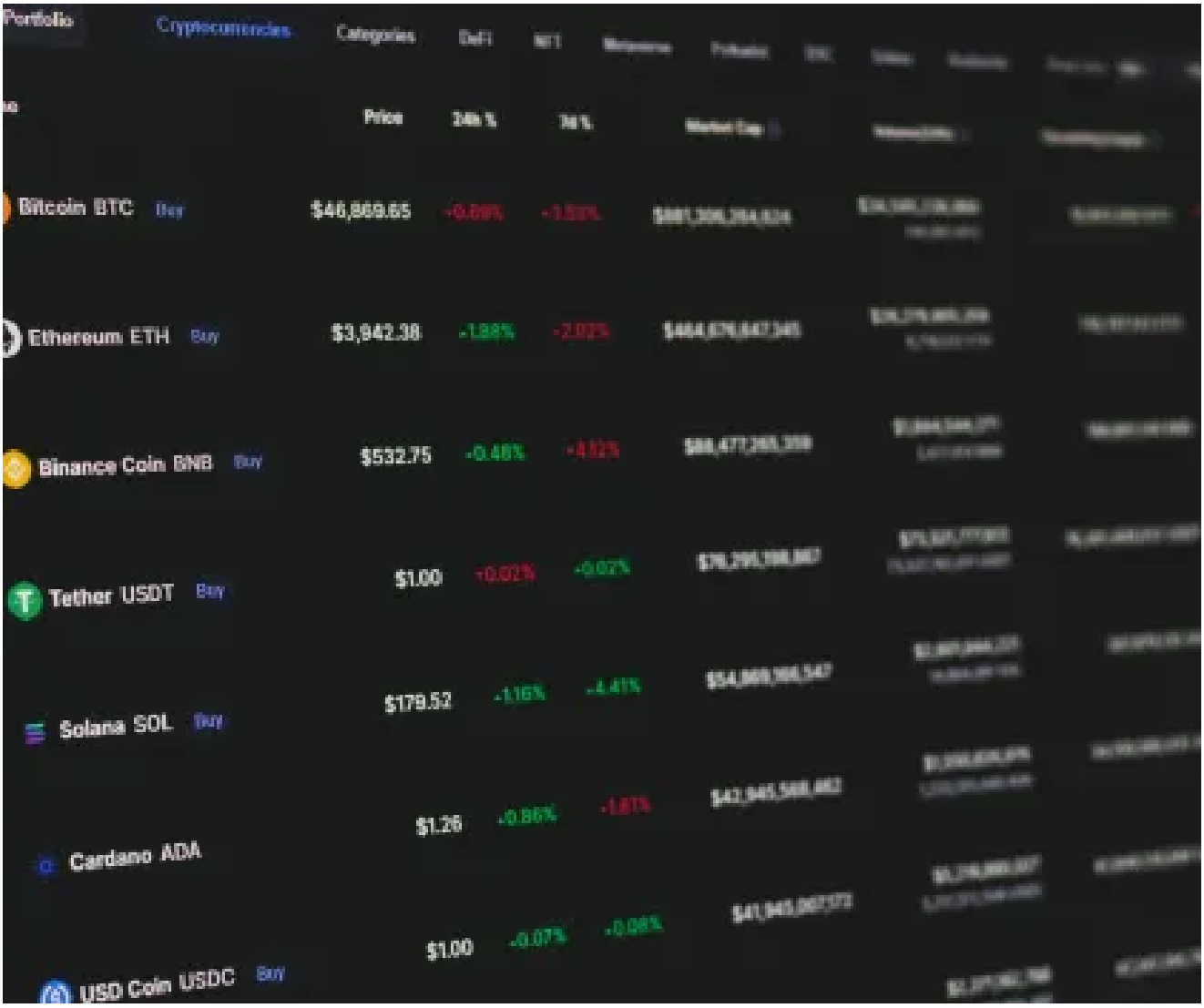
Evolution of Lazarus ‘FudModule - no longer (stand)alone’

gendigital.com/blog/news/innovation/lazarus-fudmodule-v3



In early June, we discovered a sample that was exploiting a new zero-day vulnerability within Winsock driver (**CVE-2024-38193**) to achieve local privilege escalation to deploy a new version of FudModule rootkit. We determined that the sample was part of a Lazarus Group operation that was targeting potentially sensitive industries such as aerospace and cryptocurrency engineering to gain access to their corporate networks.

CVE-2024-38193 is a use-after-free vulnerability in the `AFD.sys` driver which is responsible for kernel-mode support for the Windows socket (Winsock) interface used in network communication. This flaw allowed attackers to tamper with certain values in kernel

structures, leading to read/write access which was leveraged by FudModule to execute a Direct Kernel Object Manipulation technique. This vulnerability is present on Windows 11 Version 23H2 (and earlier) and Windows 10.

The FudModule itself has also undergone a significant evolution. In addition to its usual arsenal, it started to disable crash dumps to further complicate any investigation. It is also more closely tied to the payload it aims to protect, which it injects into a process secured by Protected Process Light. These extensive changes led us to a decision to denote this version as FudModule v3.0.

Initial access

Previous versions of FudModule (v2.0) were delivered by Kaolin RAT, utilizing another zero-day exploit (**CVE-2024-21338**) to gain read/write access to the kernel memory. Unfortunately, we've been unable to determine how **CVE-2024-38193** was delivered, along with FudModule v3.0, to the victim's device.

On August 19, 2024, Microsoft identified a North Korean threat actor exploiting a Chromium remote-code-execution (RCE) zero-day vulnerability (**CVE-2024-7971**) which was based on a type of confusion issue in the V8 JavaScript engine and WebAssembly engine. Microsoft noted that Citrine Sleet used Chromium exploit to deploy a FudModule rootkit, nevertheless, they did not specify more technical details on the rootkit. The RCE vulnerability was used to deploy a shellcode containing another exploit (**CVE-2024-38106**) that was used to escape Chromium's sandbox and deploy the downloaded FudModule rootkit into the memory.

Due to the timing of this attack and a close presumed relation between Citrine Sleet and Lazarus group, we presume that the FudModule v3.0 and Winsocks exploit (**CVE-2024-38193**) may have been delivered in a very similar manner.

FudModule Rootkit delivery

In the first version of the FudModule, referred to as FudModule v1.0, the attacker required the ability to modify kernel memory structures, potentially resulting in the deactivation of Windows monitoring features. To achieve this, they employed the Bring Your Own Vulnerable Driver (BYOVD) technique, a well-known method where attackers load a legitimate, signed, and vulnerable kernel driver to bypass the Driver Signature Enforcement (DSE) policy. The attacker then used this known vulnerability to obtain read/write access to the kernel space.

They accomplished this by loading vulnerable drivers such as `dbutil_2_3.sys` (Dell driver) and `ene.sys` (RGB lightning control driver), exploiting them using N-day vulnerabilities. This is a straightforward method, as there are numerous publicly available proof-of-concept exploits for various vulnerabilities. However, the downside is that the attacker must drop the vulnerable driver onto the filesystem and load it with administrator privileges.

The second version of the FudModule, referred to as FudModule v2.0, was discovered by Gen. In this case, the attackers exploited a zero-day vulnerability (**CVE-2024-21338**) in the Windows' `appid.sys` (Application Identity driver). The key advantage of this choice is that this driver is already present on Windows 10 and Windows 11 by default, eliminating the need to drop "your" own vulnerable driver.

However, a challenge with this approach is that the attacker needed administrator privileges to send the vulnerable IOCTL code (`0x22A018`) to the `appid.sys` driver. For more technical details about this exploit, see our previous blogpost on this vulnerability.

In this new instance, the attackers discovered a way to exploit a vulnerability in the default driver without requiring administrator privileges to interact with it. They identified a zero-day vulnerability in the `AFD.sys` (Winsocks driver) to achieve read/write (R/W) access in kernel space. After triggering the vulnerability in `AFD.sys`, the attackers first had to gain increment primitive, crafting a specific kernel address to achieve temporary R/W primitive. With these temporary primitives, they were able to corrupt the `PreviousMode` field of the current thread in the `KTHREAD` structure. By modifying the `PreviousMode`, they could bypass kernel mode checks within system calls such as `NtReadVirtualMemory` or `NtWriteVirtualMemory`, and stealthily deploy FudModule v3.0.

FudModule rootkit

The FudModule v1.0 was discovered by the ESET and AnhLab in September 2022. They described 7 techniques used to disarm security solutions and monitoring tools.In June of last year, our discovery of the FudModule 2.0 lead to a sample featuring nine techniques, out of which four were new and three were improved, with another two remaining unchanged.

In FudModule 3.0, eight techniques from the previous version remain unchanged, and one technique was removed in favour of two new techniques. The unchanged functions, are:

- Disable a security solution ability to monitor registry operations
- Disable object callbacks routine which is which is used to execute a custom code response to thread, process, and desktop handle operations.
- Disable process, thread and image kernel callbacks
- Disabling all monitoring and antivirus file system minifilters
- Disable network traffic filtering (happens only if Kaspersky drivers are present and at the same time Symantec/Broadcom drivers are absent).
- Two approaches to Disabling all system loggers in a more generic way in Event Tracing for Windows (ETW)
- Disable monitoring of `MsMpEng.exe` process (Microsoft Defender) via handle table manipulation. For `asdsvc.exe` (AhnLab Smart Defense Service), the rootkit clears the `_EPROCESS.Protection` byte, reducing it to a standard, non-protected process after the modification.

The function responsible for disabling image verification callbacks is no longer present in FudModule v3.0.

Now, we'll dive into the two new techniques included in FudModule v3.0.

```
if ( exploit(context_struct) )
{
  msg(context_struct, "enable_god_mode passed.");
  disable_crashdump(context_struct);
  clear_registry_callback(context_struct);
  disable_minifilters(context_struct);
  disable_etw_system_loggers(context_struct);
  if ( clear_callback_routine_object(context_struct) == 1
    && process_image_thread_callbacks(context_struct) == 1
    && wfp_callouts(context_struct) == 1
    && etw_provider_guids(context_struct) == 1
    && suspend_microsoft(context_struct) == 1
    && get_services_and_disable_anhlab(context_struct) == 1
    && inject_first_shellcode_to_services(context_struct) == 1
    && next_stage )
  {
    uint32_success = inject_second_shellcode_to_msiexec(context_struct, next_stage);
    msg_str = "Success.";
    ret = uint32_success;
    if ( !uint32_success )
      msg_str = "remote_exec failed.";
    msg(context_struct, msg_str);
  }
  cleanup(context_struct);
  LocalFree(context_struct);
  return ret;
}
```

Figure 1: The rootkit's main function, Fudmodule 3.0

Disabling crash dump

In FudModule v3.0 rootkit, disabling a crash dump is the first technique to be executed after a successful exploitation. Since FudModule v3.0 directly manipulates kernel structures, which could potentially cause a system crash, it employs a method to prevent the creation of crash dump files. Crash dump files capture the system's memory at the time of an error, including loaded drivers, running processes, and active kernel data. If a crash dump file caused by the rootkit were saved, it could provide leads to the exploit during the crash investigation, which could expose the zero-day before gaining foothold on the device.

To disable crash dump, the rootkit first locates the virtual address of the data sections of the `crashdmp.sys` driver and `ntoskrnl.exe` (kernel image) in memory. It then scans for pointer in the data section of `ntoskrnl.exe`; more specifically it is looking for a pointer pointing to the data section of `crashdmp.sys` which is a global pointer named CrashdmpDumpBlock. It then zeroes out this pointer which effectively prevents the system from generating a crash dump.

```
if ( DataSectionOfNT )
{
  VirtualAddrOfDataSectionNT_ = VirtualAddrOfDataSectionNT;
  NtWriteVirtualMemory(DataSectionOfNT, VirtualAddrOfDataSectionNT, *&VirtualSizeOfDataSegmentNT_);
  it = 0;
  if ( VirtualSizeOfDataSegmentNT_ )
  {
    for ( i = DataSectionOfNT_; *i <= VirtualAddrOfDataSectionOfCD || *i >= VirtualAddrOfDataSectionOfCD + VirtualSizeOfDataSegmentCD; ++i )
    {
      it += 8;
      if ( it >= VirtualSizeOfDataSegmentNT_ )
      {
        LocalFree(DataSectionOfNT_);
        return 0LL;
      }
    }
    addrOfCrashdmpDumpBlock = &VirtualAddrOfDataSectionNT_[it];
  }
  LocalFree(DataSectionOfNT_);
  return addrOfCrashdmpDumpBlock;
```

Figure 2: Loop searching for a searched pointer in the data section of ntoskernel

Shellcode Injections

The second technique, shellcode injection, is rather peculiar in the sense that it makes a shift from FudModule v2.0 as a final payload to FudModule v3.0 used as a stager for another payload. The process involves two consecutive shellcode injections into different processes: one into `services.exe` and the other into `msiexec.exe`.

Initially, memory pages are allocated within these processes, their permissions modified to allow execution, and `_EPROCESS.MitigationFlags` is cleared to avoid instability if the handle table manipulation goes awry.

```
strcpy(str_msiexec, "msiexec.exe");
MEMORY[0x155D4D70000] = 0;
CreateProcess = api_resolution(0xB1FC7F66LL, 0xF390B59FLL);
memset(&startupInfo, 0LL, 104LL);
memset(&procInfo, 0LL, 24LL);
startupInfo.cb = 104;
result = CreateProcess(str_msiexec, 0LL, 0LL, 0LL, 0, 0, 0LL, 0LL, &startupInfo, &procInfo);
MEMORY[0x155D4D70000] = procInfo.dwProcessId;
return result;
```

Figure 3: The first shellcode, that launch new instance of ”msiexec.exe”

In the case of `services.exe` injection, FudModule uses a non-protected process. This should protect it from acquiring a handle to processes protected by PPL (Protected Process Light) such as `services.exe`, `wininit.exe,` or `csrss.exe`. Nevertheless, FudModule exploits the kernel Read/Write primitive to gain direct access to the handle table. This way, it can craft a custom handle table entry, gaining control over both referenced object and its access bits, effectively bypassing PPL. Then `msiexec.exe` process is spawned from within this context.

```
xor_key = 0xE3;
resolve_api(&api);
s = *enc_payload;
memset = api.memset;
image_dos_header = api.LocalAlloc(64LL, s);
image_dos_header_ = image_dos_header;
if ( s )
{
  payload = image_dos_header;
  v8 = enc_payload - image_dos_header;
  size = s;
  do
  {
    *payload = xor_key ^ payload[v8 + 4];
    ++payload;
    --size;
    xor_key *= 13;
  }
  while ( size );
}
resolve_import_table_and_executes_entrypoint(&api, image_dos_header, s);
api.LocalFree(image_dos_header_);
```

Figure 4: The second shellcode, that is decrypting the payload

Previously in v2.0, FudModule used a dummy thread to insert an entry into the handle table and subsequently modified the referenced object to target specific processes (in order to disable security solutions). In v3.0, a more sophisticated approach is used: FudModule uses `NtDuplicateObject` function to duplicate the current process handle, creating another entry in the handle table. This new entry points to the `_OBJECT_HEADER` of the target `_EPROCESS` – in our case `services.exe`.

The second injection, targeting the `msiexec.exe` spawned by the first injection, simply aims to decrypt and prepare a payload before executing it. This payload is passed as an argument by the initial 0-day containing shellcode.

This development marks an interesting shift in the FudModule's functionality. Instead of being the final payload by itself, it serves as a loader that abuses its access into PPL-protected processes to improve stealthiness of the final payload and hinder detection and tracing.

Conclusion

The Lazarus cybercriminal group, one of the long-standing APT (Advanced Persistent Threat) groups, has once again surprised the cybersecurity community. Lazarus continues to aggressively utilize new zero-day vulnerabilities, targeting not just operating systems but also

browsers to effectively deliver their payloads. This time, they even went one step further by going from BYOVD to directly exploiting system drivers.

Since the discovery of the FudModule, it has been clear that this rootkit is under active development. Its aim appears to be achieving long-term persistence and improved stealth capabilities to bypass built-in security features and evade detection from security vendors. Its latest evolution seems to shift its purpose from a standalone rootkit for disarming security solutions to a more generic rootkit that also actively protects the associated payload. Despite our extensive research, we have not yet uncovered the exact method used to deliver the Local Privilege Escalation (LPE) exploit alongside the FudModule v3.0 rootkit. However, based on the research from Microsoft, we now believe that the LPE exploit could have been delivered through a Remote Code Execution (RCE) vulnerability in Google Chrome.

Indicators of Compromise (IoCs)

A YARA rule for the FudModule v3.0 is available at github.com/avast/ioc

Luigino Camastra

Author at Avast Threat Labs