Breaking Down Linux.Gomir: Understanding this Backdoor’s TTPs | Splunk

By Splunk Threat Research Team, Teoderick Contreras

Published: 2024-07-15 · Archived: 2026-04-05 14:34:58 UTC

A supply chain attack is a prominent "Initial Access" tactic employed by malware authors and Advanced Persistent Threat (APT) groups to gain a foothold on their targeted hosts or systems. This method involves compromising a third-party service or software that is trusted by the target, thereby injecting malicious code into legitimate software updates or distributions.

This incident underscores the critical importance of securing the software supply chain, as even trusted software can become a vector for sophisticated cyber attacks if it is compromised at any point in its distribution or update process.

In February 2024, S2W researchers and Ahnlab Security Intelligence Center (ASEC) reported a notable campaign conducted by the Kimsuky group. This APT group exploited supply chain vulnerabilities in various software solutions, specifically targeting TrustPKI and NX_PRNMAN. The compromised software packages were embedded with malicious payloads, including the GoBear backdoor. These embedded malware components enabled the attackers to infiltrate and compromise targeted hosts, facilitating unauthorized access and control.

In May 2024, Symantec released a blog uncovering a Linux version of the GoBear backdoor, which they named Linux.Gomir. This Linux variant shares several similarities in terms of code and behavior with its predecessors.

In this blog, the Splunk Threat Research Team provides an analysis of Linux.Gomir to help security analysts, blue teamers and Splunk customers defend against this threat. Below, we will review associated Tactics, Techniques and Procedures (TTPs); Atomic Tests you can use to simulate Linux.Gomir behaviors; and security content you can use to help detect this threat.

Linux.Gomir TTPs

Non-Standard Encoding (T1132.002)

Linux.Gomir generates a unique beacon or infection ID for the compromised host, which it then sends to its Command and Control (C2) server. This beacon ID is created by taking the first 10 characters of the MD5 hash derived from the username and hostname of the infected host. The beacon or infection ID is crucial for the attackers to uniquely identify and manage the compromised machines within their network. Figure 01 shows screenshots of the code that retrieves the username and hostname to generate this beacon ID.

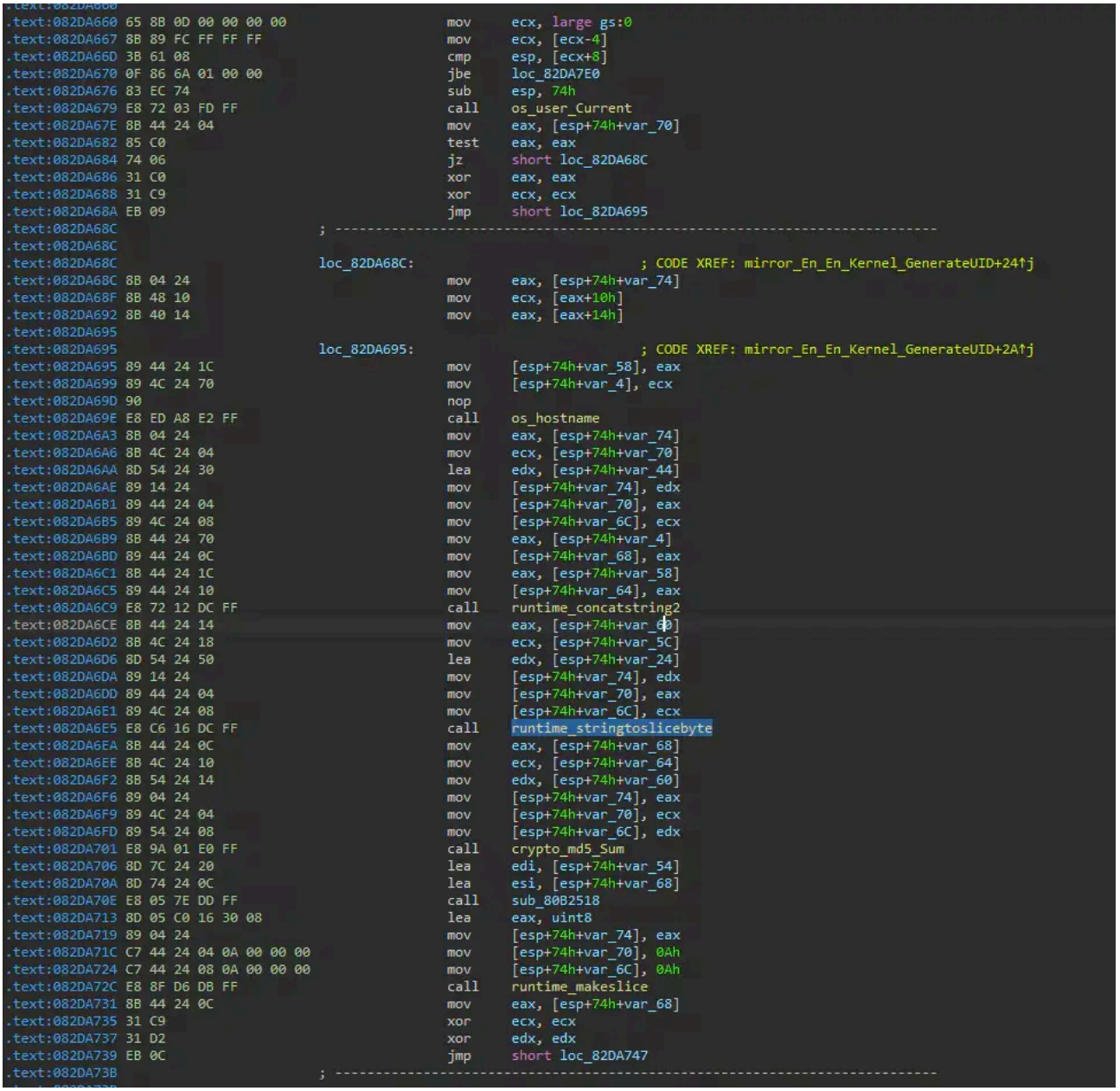


Figure 01: Generate Beacon ID

Figure 02 illustrates a simple HTTP POST network traffic instance of this backdoor malware as it communicates with its C2 server to request a backdoor command. The HTTP POST request contains a body that includes the infection ID generated on the compromised host. This infection ID allows the C2 server to uniquely identify and interact with the specific infected machine, ensuring precise command execution.

Below is the structure of the HTTP POST request body:

```
a><random_generated_string>=2&b<random_generated_string>=g-<beacon_id>1&c<random_generated_string>=
```

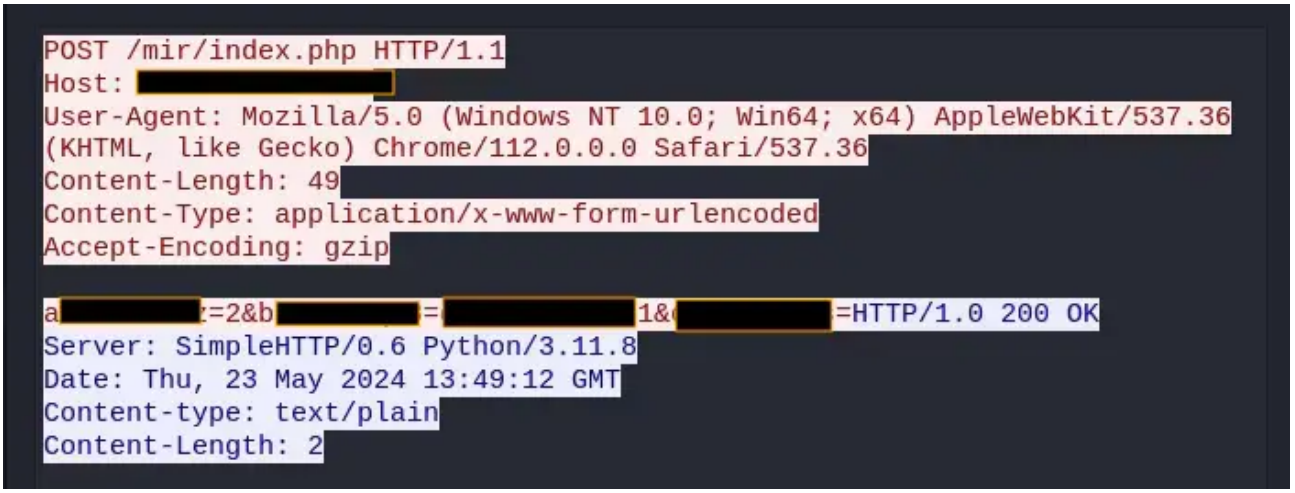
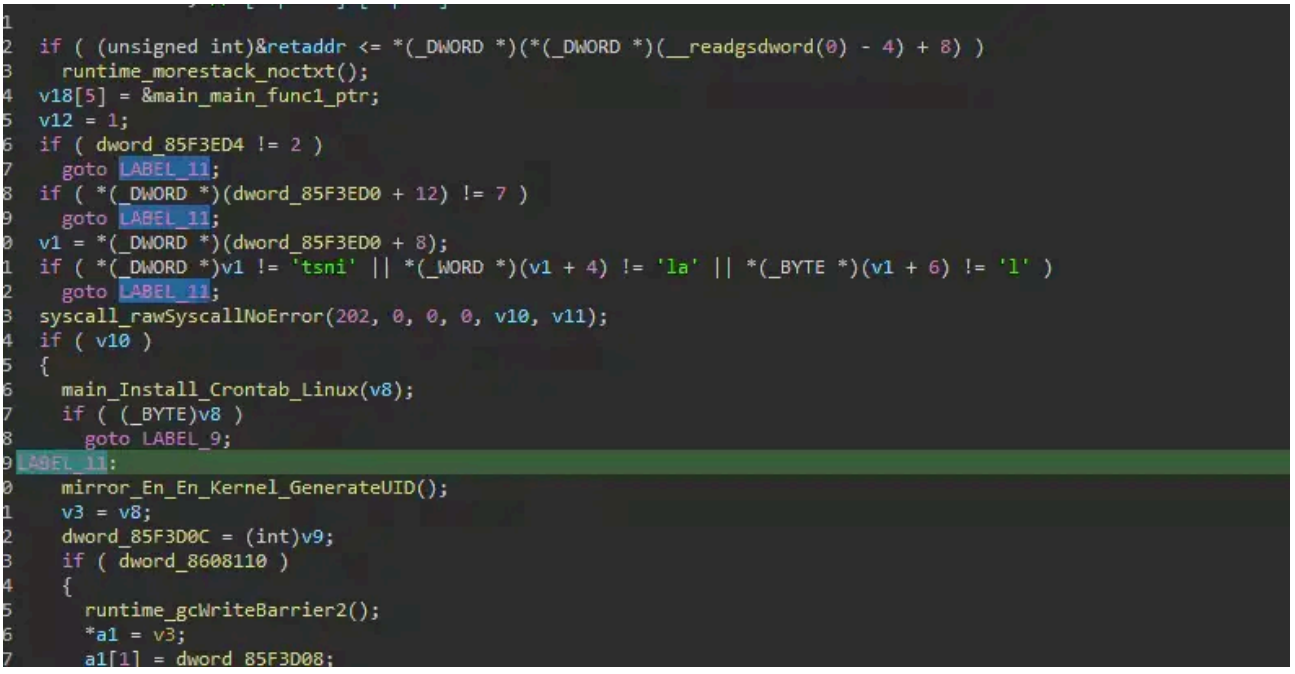


Figure 02: HTTP POST Request Body

Scheduled Task/Job: Cron (T1053.003)

Linux.Gomir includes a parameter named 'Install.' When this parameter is used, the embedded malware component executes its full range of malicious functions. These functions include generating an infection ID, dropping a copy of itself, persistence, installing services, and enabling its backdoor capabilities. The service installation and backdoor capabilities facilitate ongoing control and exploitation of the infected system.



```
if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)(__readgsdword(0) - 4) + 8) )
  runtime_morestack_noctxt();
v18[5] = &main_main_func1_ptr;
v12 = 1;
if ( dword_85F3ED4 != 2 )
  goto LABEL_11;
if ( *(_DWORD *)(dword_85F3ED0 + 12) != 7 )
  goto LABEL_11;
v1 = *(_DWORD *)(dword_85F3ED0 + 8);
if ( *(_DWORD *)v1 != 'tsni' || *(_WORD *)(v1 + 4) != 'la' || *(_BYTE *)(v1 + 6) != 'l' )
  goto LABEL_11;
syscall_rawSyscallNoError(202, 0, 0, 0, v10, v11);
if ( v10 )
{
  main_Install_Crontab_Linux(v8);
  if ( (_BYTE)v8 )
    goto LABEL_9;
LABEL_11:
  mirror_En_En_Kernel_GenerateUID();
  v3 = v8;
  dword_85F3D0C = (int)v9;
  if ( dword_8608110 )
  {
    runtime_gcWriteBarrier2();
    *a1 = v3;
    a1[1] = dword_85F3D08;
```

Figure 03: “Install” Parameter

Figure 04 illustrates the code snippet demonstrating how Linux.Gomir sets up a crontab entry as part of its persistence mechanism.

```
.text:082DE5F4 C7 44 24 18 01 00 00 00          mov     [esp+0A0h+var_88], 1
.text:082DE5FC E8 DF D3 DB FF                   call    runtime_concatstring3
.text:082DE601 8B 44 24 20                      mov     eax, [esp+0A0h+var_80]
.text:082DE605 89 44 24 2C                      mov     [esp+0A0h+var_74], eax
.text:082DE609 8B 4C 24 1C                      mov     ecx, [esp+0A0h+var_84]
.text:082DE60D 89 4C 24 78                      mov     [esp+0A0h+var_28], ecx
.text:082DE611 8D BC 24 84 00 00 00             lea     edi, [esp+0A0h+var_1C]
.text:082DE618 31 C0                            xor     eax, eax
.text:082DE61A E8 0B 3A DD FF                   call    sub_80B202A
.text:082DE61F 8D 05 CA 2D 34 08                lea     eax, aC         ; "-c"
.text:082DE625 89 84 24 84 00 00 00             mov     [esp+0A0h+var_1C], eax
.text:082DE62C C7 84 24 88 00 00 00 02+         mov     [esp+0A0h+var_18], 2
.text:082DE62C 00 00 00
.text:082DE637 8D 05 67 39 34 08                lea     eax, aCrontab   ; "crontab"
.text:082DE63D 89 84 24 8C 00 00 00             mov     [esp+0A0h+var_14], eax
.text:082DE644 C7 84 24 90 00 00 00 07+         mov     [esp+0A0h+var_10], 7
.text:082DE644 00 00 00
.text:082DE64F 8D 05 CC 2D 34 08                lea     eax, asc_8342DCC ; "-l"
.text:082DE655 89 84 24 94 00 00 00             mov     [esp+0A0h+var_C], eax
.text:082DE65C C7 84 24 98 00 00 00 02+         mov     [esp+0A0h+var_8], 2
.text:082DE65C 00 00 00
.text:082DE667 8D 05 60 39 34 08                lea     eax, aBinSh     ; "/bin/sh"
.text:082DE66D 89 04 24                         mov     [esp+0A0h+var_A0], eax
.text:082DE670 C7 44 24 04 07 00 00 00          mov     [esp+0A0h+var_9C], 7
.text:082DE678 8D 84 24 84 00 00 00             lea     eax, [esp+0A0h+var_1C]
.text:082DE67F 89 44 24 08                      mov     [esp+0A0h+var_98], eax
.text:082DE683 C7 44 24 0C 03 00 00 00          mov     [esp+0A0h+var_94], 3
.text:082DE68B C7 44 24 10 03 00 00 00          mov     [esp+0A0h+var_90], 3
.text:082DE693 E8 88 79 FF FF                   call    os_exec_Command
.text:082DE698 8B 44 24 14                      mov     eax, [esp+0A0h+var_8C]
.text:082DE69C 89 04 24                         mov     [esp+0A0h+var_A0], eax
.text:082DE69F E8 4C A4 FF FF                   call    os_exec__Cmd_Output
.text:082DE6A4 8B 44 24 10                      mov     eax, [esp+0A0h+var_90]
.text:082DE6A8 85 C0                            test    eax, eax
.text:082DE6AA 74 0A                            jz      short loc_82DE6B6
.text:082DE6AC 8B 4C 24 78                      mov     ecx, [esp+0A0h+var_28]
.text:082DE6B0 8B 44 24 2C                      mov     eax, [esp+0A0h+var_74]
.text:082DE6B4 EB 46                            jmp     short loc_82DE6FC
```

Figure 04: Sample of code demonstrating the process to create a crontab entry

The process begins by creating a file named "cron.txt" in the current working directory, which contains an entry pointing to the malware's file path.

```
@reboot <gomir_process_file_path>
```

Next, the malware attempts to list all existing crontab entries on the compromised host by executing the following command:

```
/bin/sh -c crontab -l
```

The output of this command, which lists the current crontab entries, is appended to the "cron.txt" file. Finally, the malware updates the crontab configuration by executing the following crontab command:

```
/bin/sh -c crontab cron.txt
```

Figure 05 shows the value of the existing crontab to our test lab after we execute Linux.Gomir with higher privilege.

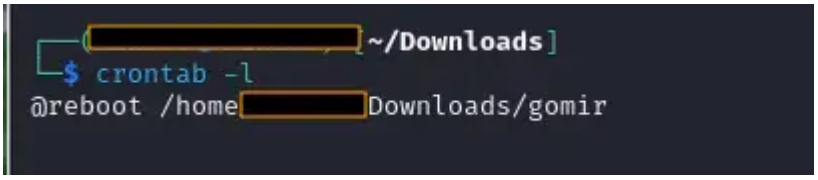


Figure 05: list crontab

Install Services

To maintain persistence and ensure execution with elevated privileges upon reboot, Linux.Gomir installs itself as a system service. Initially, it copies its executable to the **"/var/log/syslogd"** directory. Following this, it creates a service configuration file named **"syslogd.service"** in the **"/etc/systemd/system/"** directory. This service file is configured to point to the copied malware executable in **"/var/log/syslogd,"** effectively setting up the malware to run as a system service.

Lastly, to make sure that the service will smoothly install on the compromised host, it will enable and start the created service file using the following command:

```
 /bin/sh -c systemctl daemon-reload
/bin/sh -c systemctl reenable syslogd
/bin/sh -c systemctl start syslogd
```

Figure 06 contains screenshots of the code demonstrating how this malware executes the described technique.

```
.text:082DE31F 0F 85 75 01 00 00                 jnz     loc_82DE49A
.text:082DE325 89 54 24 20                       mov     [esp+34h+var_14], edx
.text:082DE329 89 4C 24 24                       mov     [esp+34h+var_10], ecx
.text:082DE32D 8D 05 C7 71 34 08                 lea     eax, aVarLogSyslogd ; "/var/log/syslogd"
.text:082DE333 89 44 24 08                       mov     [esp+34h+var_2C], eax
.text:082DE337 C7 44 24 0C 10 00 00 00           mov     [esp+34h+var_28], 10h
.text:082DE33F E8 6C B4 FC FF                    call    mirror_common_Copy_File
.text:082DE344 8B 44 24 10                       mov     eax, [esp+34h+var_24]
.text:082DE348 85 C0                             test    eax, eax
.text:082DE34A 0F 85 41 01 00 00                 jnz     loc_82DE491
.text:082DE350 8D 05 C7 71 34 08                 lea     eax, aVarLogSyslogd ; "/var/log/syslogd"
.text:082DE356 89 04 24                          mov     [esp+34h+var_34], eax
.text:082DE359 C7 44 24 04 10 00 00 00           mov     [esp+34h+var_30], 10h
.text:082DE361 C7 44 24 08 FF 01 00 00           mov     [esp+34h+var_2C], 1FFh
.text:082DE369 E8 D2 47 E2 FF                    call    os_chmod
.text:082DE36E 8B 44 24 0C                       mov     eax, [esp+34h+var_28]
.text:082DE372 85 C0                             test    eax, eax
.text:082DE374 0F 85 0E 01 00 00                 jnz     loc_82DE488
.text:082DE37A 90                                nop
.text:082DE37B 8D 05 7C F7 34 08                 lea     eax, aEtcSystemdSyst ; "/etc/systemd/system/syslogd.service"
.text:082DE381 89 04 24                          mov     [esp+34h+var_34], eax
.text:082DE384 C7 44 24 04 23 00 00 00           mov     [esp+34h+var_30], 23h ; '#'
.text:082DE38C C7 44 24 08 42 02 00 00           mov     [esp+34h+var_2C], 242h
.text:082DE394 C7 44 24 0C B6 01 00 00           mov     [esp+34h+var_28], 1B6h
.text:082DE39C E8 BF 3E E2 FF                    call    os_OpenFile
.text:082DE3A1 8B 44 24 14                       mov     eax, [esp+34h+var_20]
.text:082DE3A5 85 C0                             test    eax, eax
.text:082DE3A7 0F 85 D2 00 00 00                 jnz     loc_82DE47F
.text:082DE3AD 8B 44 24 10                       mov     eax, [esp+34h+var_24]
.text:082DE3B1 C7 44 24 28 00 00 00 00           mov     [esp+34h+var_C], 0
.text:082DE3B9 C7 44 24 2C 00 00 00 00           mov     [esp+34h+var_8], 0
.text:082DE3C1 8D 0D C0 E4 2D 08                 lea     ecx, main_Install_Service_Linux_func1
.text:082DE3C7 89 4C 24 28                       mov     [esp+34h+var_C], ecx
.text:082DE3CB 89 44 24 2C                       mov     [esp+34h+var_8], eax
.text:082DE3CF 8D 4C 24 28                       lea     ecx, [esp+34h+var_C]
.text:082DE3D3 89 4C 24 30                       mov     [esp+34h+var_4], ecx
.text:082DE3D7 C6 44 24 1F 01                    mov     [esp+34h+var_15], 1
.text:082DE3DC 8D 0D 4B 8F 35 08                 lea     ecx, aUnitAfterNetwo ; "\n[Unit]\nAfter=network.target\nDescrip"...
.text:082DE3E2 F7 D9                             neg     ecx
.text:082DE3E4 90                                nop
.text:082DE3E5 81 F9 9A 00 00 00                 cmp     ecx, 9Ah
.text:082DE3EB 0F 82 B2 00 00 00                 jb      loc_82DE4A3
.text:082DE3F1 89 04 24                          mov     [esp+34h+var_34], eax
.text:082DE3F4 8D 05 4B 8F 35 08                 lea     eax, aUnitAfterNetwo ; "\n[Unit]\nAfter=network.target\nDescrip"...
.text:082DE3FA 89 44 24 04                       mov     [esp+34h+var_30], eax
.text:082DE3FE C7 44 24 08 9A 00 00 00           mov     [esp+34h+var_2C], 9Ah
.text:082DE406 C7 44 24 0C 9A 00 00 00           mov     [esp+34h+var_28], 9Ah
.text:082DE40E E8 BD 3A E2 FF                    call    os__File_Write
.text:082DE413 8D 05 8C 9F 34 08                 lea     eax, aSystemctlDaemo ; "systemctl daemon-reload"
.text:082DE419 89 04 24                          mov     [esp+34h+var_34], eax
.text:082DE41C C7 44 24 04 17 00 00 00           mov     [esp+34h+var_30], 17h
.text:082DE424 E8 67 FA FF FF                    call    mirror_En_En_Kernel_RunShell
.text:082DE429 8D 05 18 B4 34 08                 lea     eax, aSystemctlReena ; "systemctl reenable syslogd"
.text:082DE42F 89 04 24                          mov     [esp+34h+var_34], eax
.text:082DE432 C7 44 24 04 1A 00 00 00           mov     [esp+34h+var_30], 1Ah
.text:082DE43A E8 51 FA FF FF                    call    mirror_En_En_Kernel_RunShell
.text:082DE43F 8D 05 A3 9F 34 08                 lea     eax, aSystemctlStart ; "systemctl start syslogd"
.text:082DE445 89 04 24                          mov     [esp+34h+var_34], eax
.text:082DE448 C7 44 24 04 17 00 00 00           mov     [esp+34h+var_30], 17h
.text:082DE450 E8 3B FA FF FF                    call    mirror_En_En_Kernel_RunShell
.text:082DE455 8B 44 24 24                       mov     eax, [esp+34h+var_10]
.text:082DE459 89 04 24                          mov     [esp+34h+var_34], eax
.text:082DE45C 8B 44 24 20                       mov     eax, [esp+34h+var_14]
.text:082DE460 89 44 24 04                       mov     [esp+34h+var_30], eax
.text:082DE464 E8 E7 50 E2 FF                    call    os_Remove
```

Figure 06: Sample of code used to execute the install services technique

Figure 07 shows the content of the .service file created by this malware for the copy of itself named as **“syslogd”**.

```
└─# cat /etc/systemd/system/syslogd.service

[Unit]
After=network.target
Description=syslogd

[Service]
ExecStart=/bin/sh -c "/var/log/syslogd"
Restart=always

[Install]
WantedBy=multi-user.target
```

Figure 07: Content of .service file

Command and Control (TA0011)

Finally, the malware executes a function responsible for communicating with its C2 server. Unfortunately, the C2 server was already down at the time of writing. This function decrypts incoming commands from the server and encrypts the results of the operations performed on the compromised host.

Figure 08 displays a brief snippet of the function code that processes operations after decoding and decrypting its backdoor commands. The shown code snippet pertains to three backdoor operations that retrieve the process's current working directory, get the directory size and collect system information such as CPU details, memory statistics, and network information.

```
else
{
  operation_0 = *(_BYTE *)(operation + 1);
  if ( operation_0 > '4' )
  {
    if ( operation_0 > '6' )
    {
      switch ( *(_WORD *)operation )
      {
        case '70':
          mirror_En_En_Kernel_Process_Where(a2, a3);
          v3 = v22;
          break;
        case '80':
          runtime_slicebytetostring(0, (unsigned __int8 *)(operation + (((2 - v19) >> 31) & 2)), v16 - 2);
          mirror_En_En_Kernel_Process_Dirsize(a1, a2, a3, v16, v19);
          v3 = v22;
          break;
        case '90':
          mirror_En_En_Kernel_Process_GetInfo(a2, a3);
          v3 = v22;
          break;
        default:
          goto LABEL_52;
      }
    }
```

Figure 08: Part of Backdoor Function

Below are the activities we observed the three backdoor operations execute:

- Start reverse proxy client
- Upload socket list information to the C2 server
- Hibernate the Gomir client at a specific date, time, and location
- Return the command path of the Gomir client process
- Return the code page of the Gomir client process
- Return the current file path of the process
- Return the size of the targeted directory from the compromised host
- Retrieve system information such as hostname, username, CPU details, memory statistics, and network information (interface, names, addresses)
- Set up a TCP connection
- Kill or exit a process
- Retrieve the current working directory of the Gomir client process
- Change the current working directory
- Put the Gomir client to sleep

- Execute shell commands

Detecting Linux.Gomir with Splunk Security Content

By understanding Linux.Gomir's behaviors, the Splunk Threat Research Team was able to generate telemetry and datasets to develop detections designed to help defend and respond against this threat. You can find these detections below, and for our full repository of security detections, visit research.splunk.com.

Linux Adding Crontab Using List Parameter

The following analytic identifies suspicious modifications to cron jobs on Linux systems using the crontab command with list parameters. This command line parameter can be abused by adversaries to add a crontab entry for executing their malicious code on a schedule of their choice. However, it's important to note that administrators or normal users may also use this command for legitimate automation purposes, so filtering is required to minimize false positives. Identifying the modification of cron jobs using list parameters is valuable for a SOC, as it indicates potential malicious activity or an attempt to establish persistence on the system.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
 as lastTime from datamodel=Endpoint.Processes where Processes.process_name = "crontab"
 Processes.process= "* -l*" by  Processes.parent_process_name Processes.process_name
 Processes.process Processes.process_id Processes.parent_process_id Processes.dest
 Processes.user | `drop_dm_object_name(Processes)` | `security_content_ctime(firstTime)`
 | `security_content_ctime(lastTime)` | `linux_adding_crontab_using_list_parameter_filter`
```

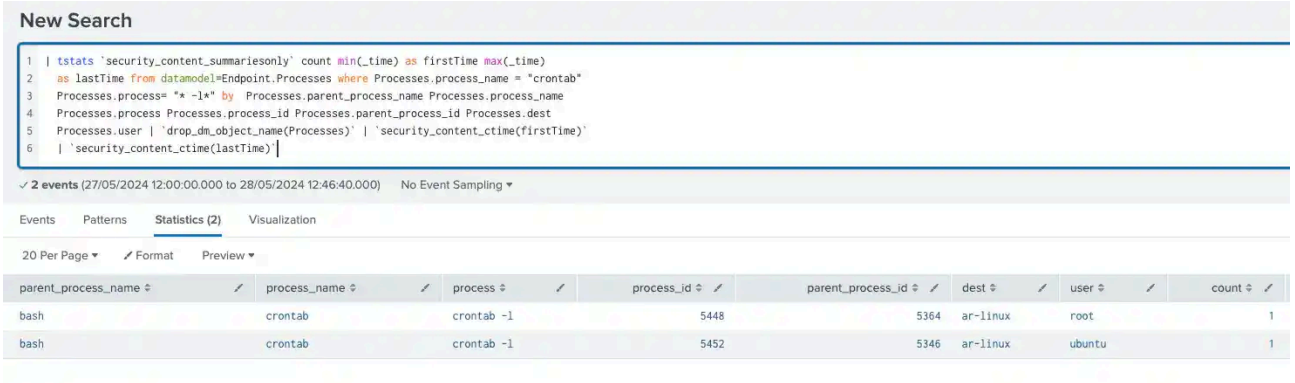


Figure 09: Detection for adding Crontab

Linux Service Restarted

The following analytic detects the restarting or re-enabling of services in the Linux platform. It focuses on the use of systemctl or service tools for executing these actions. Adversaries may leverage this technique to repeatedly execute malicious payloads as a form of persistence. Linux hosts typically start services during boot to perform background system functions. However, administrators may also create legitimate services for specific tools or applications as part of task automation. In such cases, it is recommended to verify the service path of the registered script or executable and identify the creator of the service for further validation.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
 as lastTime from datamodel=Endpoint.Processes where (Processes.process_name IN ("systemctl",  "service") OR Pr
Processes.parent_process_name
 Processes.process_name Processes.process Processes.process_id Processes.parent_process_id  Processes.process_g
```

Figure 10: Detection for Service Restarted

Linux Service Started Or Enabled

The following analytic detects the creation or enabling of services in Linux platforms, specifically using the systemctl or service tool application. This behavior is worth identifying as adversaries may create or modify services to execute malicious payloads as part of persistence. Legitimate services created by administrators for automation purposes may also trigger this analytic, so it is important to update the filter macros to remove false positives.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
 as lastTime from datamodel=Endpoint.Processes where (Processes.process_name IN ("systemctl",  "service") OR Pr
Processes.vendor_product="Microsoft Windows")
  by Processes.dest Processes.user Processes.parent_process_name  Processes.process_name Processes.process Proc
```

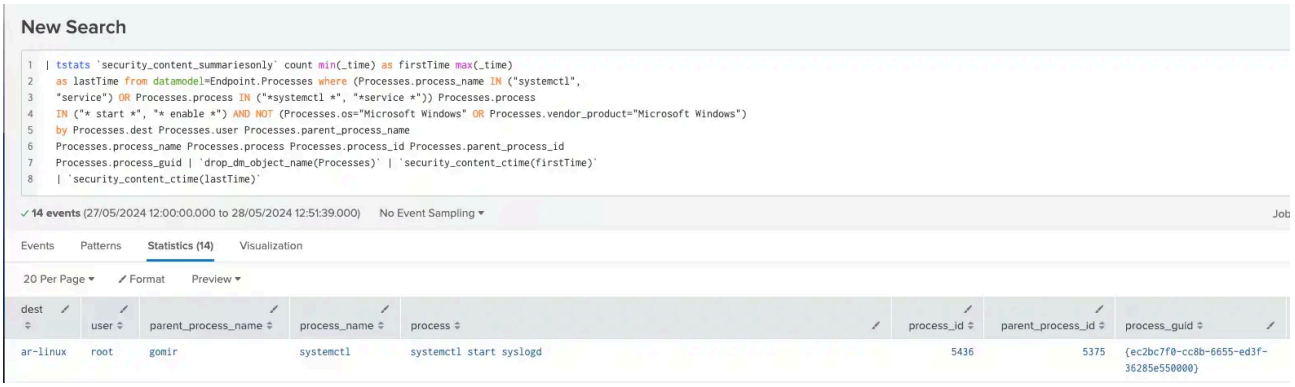


Figure 11: Detection for Service Started/Enable

Linux Service File Created In Systemd Directory

The following analytic is designed to detect suspicious file creation within the “systemd” timer directory on Linux platforms. Malicious actors can exploit this feature by embedding a “systemd” service file for persistence on the targeted or compromised host.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
 as lastTime FROM datamodel=Endpoint.Filesystem where Filesystem.file_name = *.service  Filesystem.file_path IN
"*/usr/lib/systemd/system*",
 "*/run/systemd/system*", "*~/.config/systemd/*",
"*~/.local/share/systemd/*","*/etc/systemd/user*",
 "*/lib/systemd/user*", "*/usr/lib/systemd/user*", "*/run/systemd/user*") by Filesystem.dest  Filesystem.file_
 | `linux_service_file_created_in_systemd_directory_filter`'
```

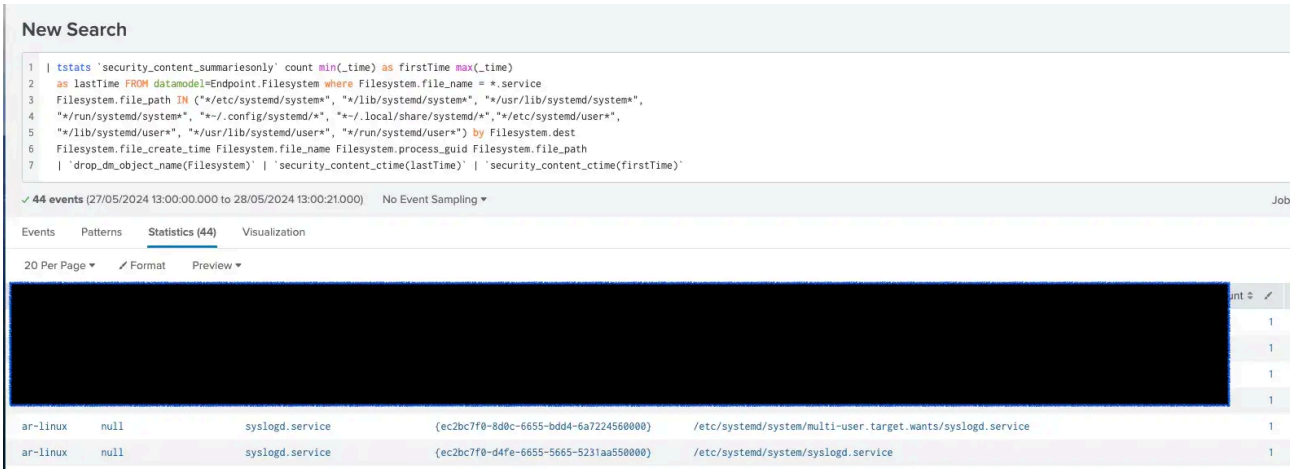


Figure 12: Detection for Created Services

Atomic Tests for Simulating Linux.Gomir

As defenders of our organization, it is essential to test the effectiveness of our defenses, including analytics, XDR, and antivirus products. This ensures they are properly configured and capable of detecting notable Linux.Gomir TTPs. In this section, we will guide you through the process of testing your detections using Atomic Red Team. These tools will help you validate and fine-tune your security measures to enhance your organization's resilience against threats.

- Create Systemd Service: This test creates a Systemd service and timer then starts and enables the timer.
- Cron - Replace crontab with referenced file: This test replaces the current user's crontab file with the contents of the referenced file. This technique was used by numerous IoT automated exploitation attacks.
- Create Systemd Service file, Enable the service, Modify and Reload the service: This test creates a “Systemd” service unit file and enables it to autostart on boot. Once the service is created and enabled, it also modifies this same service file.

IOC

Learn More

This blog helps security analysts, blue teamers and Splunk customers identify Linux.Gomir malware by enabling the community to discover related TTPs used by threat actors and adversaries. You can implement the detections in this blog using the Enterprise Security Content Updates app or the Splunk Security Essentials app. To view the Splunk Threat Research Team's complete security content repository, visit research.splunk.com.

Contributors

We would like to thank Teoderick Contreras for authoring this post and the entire Splunk Threat Research Team for their contributions.
