CoopSSLE

Technische Dokumentation

COOP, Systemtechnik Server Intel, Informatik

Ansprechpartner: Herr Patrick Stamm

Dokumentversion 1.11

Letzte Änderung: 25.08.2009

Status: abgeschlossen

InterSolutions GmbH
Hauptstrasse 23
4142 Münchenstein

Phone : +41 61 416 86 00
Fax : +41 61 416 86 01

Web : www.intersolutions.ch

Daniel Casota
IT Consultant
Direct: 061 416 86 04
Mobile: 079 467 99 20
E-Mail: daniel.casota@intersolutions.ch

Änderungsverzeichnis

Ver	Verantwortung / Autor	Bemerkungen	Datum
1.0	Daniel Casota	Erstfassung	21.03.2009
1.1	Daniel Casota	Update zur Scanletversion 2.0	13.08.2009
1.11	Daniel Casota	Update zur Scanletversion 2.01	25.08.2009

Inhaltsverzeichnis

Einleitung

Im Coop POS-Bereich wird 2009 die neue Kassenlösung der Firma GK eingeführt. Die Organisationseinheit „Systemtechnik Server Intel“ ist in diesem Zusammenhang für die Bereitstellung einer Softwareverteilungslösung verantwortlich. Neben dem geplanten Standard-Releaseverfahren für neue oder aktualisierte Softwarepakete soll zusätzlich das Scanletverfahren eingesetzt werden, welches in der alten, TSKAP-basierenden POS-Lösung für Windows-Plattformen entwickelt wurde.

Die Scanlettechnologie ermöglicht eine rasche, unabhängige Massenverteilung von Softwarepaketen, Skripts, etc., sowie das automatisierte Installieren dieser Elemente und arbeitet mit sehr wenigen, grundlegenden Voraussetzungen. Die Technik berücksichtigt keine releasespezifischen Rahmenbedingungen, Verteil- und Installationsvorschriften wie z.B. Zeitfenster oder Abhängigkeiten und ist daher nur als Verteilwerkzeug zweiter Wahl zu verstehen.

Die Firma InterSolutions GmbH wurde beauftragt, die Scanlettechnologie von der Windows-Plattform auf die Suse Linux 10 Enterprise-Plattform zu portieren.

Die vorliegende technische Dokumentation erläutert den Verarbeitungsansatz, den aktuellen Stand des Sourcecodes und den Einsatz eines Scanlets.

Die Firma InterSolutions bedankt sich bei Herrn Stamm, Herrn Kirscher und Herrn Gisi für die Berücksichtigung bei der Evaluation von Entwicklungspartnern. Sie hofft, dass die technische Portierung und das vorliegende Dokument den Erwartungen entsprechen und den Aufbau des CoopSSLE (Coop Scanlet für Suse Linux Enterprise) hinreichend erläutert.

1 Überblick

Dieses Dokument bezieht sich auf die CoopSSLE-Version 2.01

1.1 Versionierung

Die CoopSSLE-Version ergibt sich aus der Version des scanletstarter.sh-Skripts.

1.2 Voraussetzungen

Das CoopSSLE wurde bislang auf folgenden Plattformen getestet:

- Suse Linux 10 Enterprise (Coop POS Standard-Plattform)
- Open Suse 11.1

Folgende Tools werden vorausgesetzt:

- rsync
- expect

1.3 Verzeichnis- und Dateistruktur

Das CoopSSLE besteht lediglich aus einem Verzeichnis mit gewissen Unterstrukturen. Eine Installation im klassischen Sinne ist nicht notwendig, es reicht ein Hinkopieren des Verzeichnisses und eine Anpassung der Konfigurationsparameter.

```
\CoopSSLE_2.00
        list
        mailcfg.sh
        recipients
        releasenotes
        scanletstarter.sh
        scanletcfg.sh
        wishlist
        \remotescripts
                scanletjob.sh
        \remotetools
                remotemanager.sh
        \tools
                sshlogin.sh
                sshrobocopy.sh
                sshrobocopy-expectless.sh
                switcher.sh
```

Bei den schwarz markierten Elementen mit vorausgehendem „\“ handelt es sich um Verzeichnisse.

Bei den grau markierten Elementen handelt es sich um Textdateien.

Die blau markierten Elemente sind bash-Skripts.

Im Kapitel 2 *Skripterläuterungen* werden die einzelnen Dateien erläutert.

2 Skripterläuterungen

2.1 scanletcfg.sh

Ab der Version 2 des COOPSSLE sind scanletspezifische Parameter im Bash-Skript scanletcfg.sh ausgelagert. Diese können durch den Scanletentwickler entsprechend gesetzt werden.

Anbei werden die Parameter erläutert:

Keywort	Wertebereich	Bedeutung
debug	Boolean (0/1)	0: keine zusätzliche Protokollierung von scanletinternen Informationen 1: Zusätzliche Protokollierung von scanletinternen Informationen. Zusätzlich werden alle temporär erstellten Helperdateien nicht gelöscht. Defaultwert = 0
authentication_type	expect expect_create_new_ssh_rsa_key ssh_rsa_key	expect: Die SSH-Verbindung auf die Zielsysteme erfolgt mittels dem expect-Befehl. Dieser übergibt das Passwort interaktiv, jedoch automatisiert. Der Nachteil dieser Methode ist allerdings, dass mit diesem authentication_type das Scanlet nicht als cron-Job gesetzt werden kann. expect_create_new_ssh_rsa_key: Um Scanlets mittels Zertifikaten laufen lassen zu können, müssen diese zuerst erstellt und auf den Zielsystemen installiert werden. Dieser Prozess wird durch expect_create_new_ssh_rsa_key durchgeführt. Dieser authentication_type bedingt ebenfalls eine interaktive Durchführung. Zusätzlich erfolgt bei der Zertifikatserstellung das Definieren einer Passwortfrage. Diese muss leer gelassen wrden (mit ENTER setzen). ssh_rsa_key: Die Authentifizierung erfolgt komplett mittels Zertifikaten. Defaultwert = expect

Keywort	Wertebereich	Bedeutung
remote_start_method	direct cron	direct: Durch diese Option wird der Scanletjob direct beim Vorgang DISTRIBUTING ausgeführt. Es handelt sich somit um eine synchrone SSH-Verbindung vom Distributionsserver zum Zielsystem. cron: Durch diese Option wird der Scanletjob als cron-job abgelegt. Dadurch wird der DISTRIBUTING-Vorgang asynchron durchgeführt. Der Vorteil dieser Methode liegt darin, dass bei Massenverteilungen eine raschere Abwicklung erreicht werden kann.
remote_su_method	expect sudo	Die remote_su_method definiert die Methode, mit welcher auf dem Zielsystem in den root-Account gewechselt wird. expect Mit expect wird per gleichnamigem Programm interaktiv, jedoch automatisiert, in den root-Account gewechselt. sudo Hat das Skript remotemanager.sh sudo-Rechte, dann wird dieses per sudo aufgerufen. Ist dies nicht der Fall, aber dennoch die Option sudo gesetzt, dann wird die Datei /etc/sudoers per expect-Methode entsprechend angepasst. Defaultwert = direct
remote_connection_user	(string)	Name des Benutzers, welcher für ssh verwendet werden soll. Diese Angabe ist nur notwendig bei • authentication_type=expect • authentication_type= expect_create_new_ssh_rsa_key
remote_connection_password	(string)	Passwort zum Benutzer, welcher für ssh verwendet werden soll. Diese Angabe ist nur notwendig bei • authentication_type=expect • authentication_type= expect_create_new_ssh_rsa_key
Remote_root_password	(string)	Passwort zum root-Benutzer auf dem Zielsystem. Diese Angabe ist nur notwendig bei • remote_su_method=expect • sofern noch nie auf dem Zielsystem /etc/sudoers eingerichtet worden ist.

Keywort	Wertebereich	Bedeutung
maxslots	(positive Zahlen)	Anzahl der zu erstellenden Slots für die Verteilung auf dem Distributionsserver. Defaultwert = 10
waitmaxslotending	(positive Zahlen)	Anzahl Sekunden, die maximal gewartet werden soll, bis alle Slots wieder aufnahmebereit sind, ohne forciert gestoppt zu werden. Defaultwert = 600 Hinweis: Ist als remote_start_method direct gewählt, so muss der Wert für waitmaxslotending höher sein als die die Verarbeitungsdauer des Scanletjobs.
waittime	(positive Zahlen)	Wartezeit zwischen den Modi DISTRIBUTING und COLLECTING. Defaultwert = 80 Hinweis: Der Wert für waittime muss höher sein als die die Verarbeitungsdauer des Scanletjobs zusätzlich 60 Sekunden (Grund: Da cron als kleinste Auflösung nur Minuten kennt, ist für einen Job 60 Sekunden als Sicherheit einzurechnen).
cleanupremotepath	Y (anderer Stringwert)	Mit dem Wert „Y“ wird beim COLLECTING das beim DISTRIBUTING-Vorgang erstellte Scanletverzeichnis komplett gelöscht.

Hinweise zu den Kombinationen von authentication_type, remote_start_method und remote_su_method:

Solange nicht Coop-weit Zertifikate eingesetzt werden sollen, wird folgende Kombination standardmässig empfohlen:

- authentication_type = expect
- remote_start_method = cron
- remote_su_method = sudo

Folgende weitere Kombinationen sind erlaubt:

Supported (Ja/Nein)	authentication_type	remote_start_method	remote_su_method	Bemerkung
Ja	expect	direct	expect	
Nein	expect	direct	sudo	
Nein	expect	cron	expect	
Ja	expect	cron	sudo	Kann erst nach expect_create_new_ssh_rsa_key angewendet werden.
Ja	expect_create_new_ssh_rsa_key	direct	expect	
Nein	expect_create_new_ssh_rsa_key	direct	sudo	
Nein	expect_create_new_ssh_rsa_key	cron	expect	
Ja	expect_create_new_ssh_rsa_key	cron	sudo	
Nein	ssh_rsa_key	direct	expect	
Nein	ssh_rsa_key	direct	sudo	
Nein	ssh_rsa_key	cron	expect	
Ja	ssh_rsa_key	cron	sudo	Kann erst nach expect_create_new_ssh_rsa_key angewendet werden.

Hinweis:

Für das Arbeiten mittels cron-Jobs ist nur die letzte Variante (ssh_rsa_key/cron/sudo) supported.

2.2 scanletstarter.sh

Das scanletstarter.sh-Skript wird direkt durch den Benutzer oder einem cron-Job aufgerufen. Der Skriptablauf sieht wie folgendermassen aus:

- Anzeige des Headers
- Prüfung, ob der root-Kontext vorliegt und Einmalstart-Check
- Einlesen aller scanletspezifischen Parametern aus scanletcfg.sh und Logfile-Erstellung
- Initialisierung aller Konstanten und inneren Variablen
- Erstellen von Zertifikaten, sofern als authentication_type expect_create_new_ssh_rsa_key gesetzt ist
- Erstellen der definierten Anzahl Threads („Slots“)
- Abarbeiten der definierten Clientliste im Verteilmodus („Distributing“)
- Warten, bis die definierte, scanletspezifische Wartezeit abgelaufen ist
- Abarbeiten der definierten Clientliste im Abholmodus („Collecting“)
- Beenden der Threads
- Mailing
- Cleanup

In den folgenden Unterkapiteln werden aus dem Quellcode nicht offensichtliche Abläufe erläutert. Offensichtliche Abläufe sind z.B. die Anzeige des Headers oder das Abwarten einer definierten Ablaufzeit.

2.2.1 Einlesen aller scanletspezifischen Parametern aus scanletcfg.sh und Logfile-Erstellung

In der Prozedur initialization() wird das scanletcfg.sh wird aufgerufen. Damit werden alle durch den Scanletentwickler definierten Variablen gesetzt. Zusätzlich wird das Logfile erstellt.

2.2.2 Initialisierung aller Konstanten und Variablen

In der Prozedur settings() werden sämtliche Variablen gesetzt und die Voraussetzungen geprüft. Dazu gehört die Integritätsprüfung, d.h. ob alle notwendigen Dateien (z.B. switcher.sh) vorhanden sind.

2.2.3 Erstellen von Zertifikaten

Ist als authentication_type expect_create_new_ssh_rsa_key gesetzt, dann werden neue RSA-Zertifikate erstellt und für die Übermittlung auf die Zielsysteme vorbereitet.

2.2.4 Erstellen der definierten Anzahl Threads („Slots“)

Das Erstellen der Threads erfolgt über die prozedur createthreads(). Diese besteht lediglich aus einer for-Schlauf, welche bis zum maxslots-Wert jeweils die Subprozedur threadslot() aufruft. In dieser Subprozedur wird ein temporäres Verzeichnis erstellt, in welchem alle notwendigen Dateien aus dem Tools, Remotetools und remotescripts-Verzeichnis hinkopiert werden. Zudem wird eine slotspezifische Initialisierungsdatei über die Variable slotinitfile erstellt, welche sämtliche Variablen enthält und als bash-Skript aufgerufen wird. Am Schluss der Subprozedur threadslot() wird der Thread mittels folgender Zeile aufgebaut:

```
nohup "$pathslotID/$toolsdirname/$switcherfilename" "$slotinitfile" &>"$pathslotID/$slotlogfilename" &
```

Das letzte "&" führt dazu, dass das aufgerufene Bashskript switcher.sh (im temporären, jeweiligen Slotverzeichnis) im Hintergrund ausgeführt wird und das scanletstarter.sh-Skript parallel dazu weiter abgearbeitet wird.

2.2.5 Abarbeiten der definierten Clientliste im Verteilmodus („Distributing")

Die definierten in der Datei `list` definierten Clients werden in der Prozedur distribute() über ein for-Schlaufe abgearbeitet. Dabei wird jeweils die Subprozedur threadswitch() aufgerufen, welche einen freien Slot sucht und den Client diesem freien Slot zuweist. Als Inputparameter erhält diese Subprozedur einerseits die Information zum Client (FQDN oder IP-Adresse), andererseits auch den Modus (Distributing oder Collecting).

Wurden sämtliche Clients abgearbeitet, bedeutet dies aufgrund der parallelen Verarbeitung noch nicht, dass alle Threads(=Slots) wieder bereit sind. Dies wird berücksichtigt, indem am Schluss über die Subprozedur waitthreadrotatorend() der Status jedes Slots geprüft wird.

Die Prozedur threadswitch() ist lediglich eine Wrapper-Prozedur, welche die Subprozedur threadswitchprocessing() aufruft.

Der Wrapper optimiert den Ablauf: Angenommen, ein Client konnte bereits beim Verteilen (Modus=Distributing) nicht erreicht werden, dann erübrigt sich das Abholen des clientseitigen Jobergebnisses, weil gar kein Job platziert wrden konnte.

Die Prozedur threadswitchprocessing() prüft, welcher Thread(=Slot) frei ist und weist einen Client diesem freien Slot zu. Die Prüfung erfolgt auf über eine Status-Datei namens status.sh. Besitzt eine Slot eine solche Datei, dann wird damit signalisiert, dass der betreffende Slot besetzt ist. In der Prozedur wird ein Rotationsmeccano angewendet: So wird über die Variable slotpointer jeder Slot von 1 bis zu maxslots geprüft. Sind alle Slots besetzt, so wird wieder bei 1 begonnen, allerdings mit einer Zwischenpause von 2 Sekunden. Wird ein freier Slot gefunden, so wird die Status-Datei status.sh dynamisch kreiert. Enthalten sind die Informationen zum Client (FQDN oder IP-Adresse) und der Modus. Danach wird die Prozedur threadswitchprocessing() unterbrochen.

Parallel dazu beginnt nun der Slot über das Skript switcher.sh, welches regelmässig prüft, ob eine Verarbeitung vorliegt, die Datei status.sh auszulesen und entsprechend dem Modus weiterzuverarbeiten.

2.2.6 Abarbeiten der definierten Clientliste im Abholmodus („Collecting")

Die definierten in der Datei `list` definierten Clients werden in der Prozedur collect() über ein for-Schlaufe abgearbeitet. Dabei wird identisch zum Modus Distributing jeweils die Subprozedur threadswitch() aufgerufen, welche einen freien Slot sucht und den Client diesem freien Slot zuweist. Als Inputparameter erhält diese Subprozedur einerseits die Information zum Client (FQDN oder IP-Adresse), andererseits auch den Modus (Distributing oder Collecting).

Die weitere Verarbeitung durch die Subprozedur threadswitch() wurde bereits im Modus Distributing erläutert.

2.2.7 Beenden der Threads

Wurden sämtliche Clients im Modus Collecting abgearbeitet, dann können die Threads wieder beendet werden. Dies wird über die Prozedur destroythreads() vorgenommen. Das Beenden der eines Threads erfolgt gleich wie bei einer Verarbeitung ebenfalls über ein Signalisationsflag: Das switcher.sh-Skript prüft regelmässig, ob eine Datei slotendtagfilename vorliegt. Falls ja, beendet sich der Thread selber. Deswegen wird in der destroythreads()-Prozedur über eine for-Schlauf von 1 bis maxslots für jeden Slot diese Datei erstellt.

2.2.8 Mailing

In der Prozedur mail() wird sämtlichen Email-Empfängern, welche in der Datei recipients zeilenweise definiert sind, das allfällig vorhandene Errorlogfile zugesendet. Die Dateiendung wird dabei auf .csv abgeändert.

2.2.9 Cleanup

In der Prozedur cleanup() wird das temporäre Verzeichnis, in welchem sich u.a. auch alle Slots befinden, gelöscht. Diese Funktion ist derzeit noch auskommentiert.

2.3 switcher.sh

Das Bashskript switcher.sh wird im jeweiligen Slot aufgerufen und erhält als Übergabeparameter die Slotinformationen über die Datei slotinitializationfile.

In einem Intervall von einer Sekunde prüft nun das Skript zwei Dinge:

- Ist eine Datei status.sh vorhanden? Dies stellt die Signalisation dar, dass eine Verarbeitung vorgenommen werden soll. Falls dem so ist, dann wird das status.sh-Skript (über die Variable slotstatusflagfile) aufgerufen, sodass die Variablen zum Client (ipaddr) und zum Modus (modus) gesetzt werden. Danach wird die Subprozedur proceed() aufgerufen. Nach dessen Verabeitung wird das entstandene Logfile verarbeitet und das status.sh-Skript wieder gelöscht. Dies signalisiert, dass die verarbeitung eines weiteren Clients wieder möglich ist.
- Ist eine Datei end.tag (Variable slotendtagfile) vorhanden? Falls ja, so stellt dies die Signalisation dar, dass der Slot beendet werden soll.

Hinweis: Eine laufende Verarbeitung (gesetztes Slotstatusflagfile) kann grundsätzlich nicht sofort beendet werden. Ein manuell gesetztes Slotendtagfiles als Signalisation, dass der Slot zu beenden ist, wird erst verarbeitet, wenn der Slot wieder frei ist.

Die Prozedur proceed() prüft zuerst anhand des ping-Befehls, ob das System online ist. Daher darf auf dem Client-System die icmp-Nachricht für einen Ping-Feedback nicht unterdrückt sein. Im Modus Distributing wird die Subprozedur distributing() aufgerufen. Im Modus collecting wird entsprechend collecting() aufgerufen. Zusätzlich wird das durch die clientseitige Verarbeitung gesetzte Endtagfile gelöscht, allenfalls auch das komplette, clientseitige Scanletverzeichnis.

Der Ablauf im Modus Distributing sieht folgendermassen aus:

Über den rsync-Befehl, welcher im bash-Skript sshrobocopy.sh oder sshrobocopy-expectless.sh (Variable sshrobocopyfilename) gekapselt ist, werden die Verzeichnisse remotetools, remotescripts und die Slot-Initialisierungsdatei (Variable slotinitfilename) hinkopiert. Danach wird die Prozedur remotestart() aufgerufen. Konnte bis hierhin die Verarbeitung nicht erfolgreich durchgeführt werden, wird an der jeweiligen Stelle ein entsprechender Fehler ausgegeben und protokolliert.

Im Modus Collecting sieht der Ablauf folgendermassen aus:

Über rsync (sshrobocopyfilename) wird das Signalisationsfile end.tag (Variable remotedonetagfilename) vom Client her lokal in den Slot hinkopiert. Ist die Datei noch nicht vorhanden, dann bedeutet dies, dass die Verarbeitung noch nicht beendet wurde. Dies wird entsprechend mit einem Fehler Info_MachineBUSY protokolliert.

Ist die clientseitige Verarbeitung fertig, also das Signalisationsfile end.tag vorhanden, dann wird über rsync (sshrobocopyfilename) zusätzlich das clientseitige Protokoll (Variable remoteerrorlogfilename) vom Client her lokal in den Slot hinkopiert. Ist die Datei nicht

vorhanden, dann bedeutet dies, dass die Verarbeitung erfolgreich beendet wurde und somit kein Fehler vorliegt. Wurde ein Fehler gefunden, dann wird dies ins Slot-eigene Fehlerprotokoll (Variable sloterrorlogfilename) protokolliert.

Am Schluss wird die Prozedur remotestart() aufgerufen

Sowohl im Modus DISTRIBUTING als auch COLLECTING wird die Prozedur remotestart() aufgerufen. Diese führt einen su durch und wechselt in den remote_connection_user-Kontext und startet das sshlogin-Skript mit dem remotemanager.sh und slotinitfilename als Parameter.

2.4 sshrobocopy.sh

Dieses Skript kopiert mittels rsync Dateien von einer Source auf ein Target. Dabei wird die Verbindung mittels ssh verschlüsselt.

Als Automatisierungsmethode für das Login wird die expect-Methode verwendet. Zu einem späteren Zeitpunkt ist als weitere Methode die Verwendung von SSH-Keys vorgesehen.

Die Source wurde auf Basis von Foreneinträgen erstellt. Diese sind im Kommentarheader aufgelistet.

2.5 sshrobocopy-expectless.sh

Dieses Skript wird für alle nicht-expect-basierenden Methoden benötigt (zertifikatbasierend) und führt vom Inhalt her das Analge wie das sshrobocopy.sh-Skript aus.

2.6 sshlogin.sh

Das Skript sshlogin.sh erstellt eine ssh-Verbindung zu einem Client her und ruft auf diesem ein clientseitiges Skript auf. Als Automatisierungsmethode für das Login wird die expect-Methode verwendet. Zu einem späteren Zeitpunkt ist als weitere Methode die Verwendung von SSH-Keys vorgesehen.

Die Source wurde auf Basis von Foreneinträgen erstellt. Diese sind im Kommentarheader des Skripts sshrobocopy.sh aufgelistet.

2.7 sulogin.sh

Das sulogin.sh-Skript wird auf dem Client ausgeführt. Der Aufruf erfolgt im Distributing-Modus (Prozedur distributing()) durch das sshlogin.sh im switcher.sh-Skript. Das Skript wechselt den Benutzerkontext auf root und führt das remotescanlet.sh-Skript mit dem Konfigurationsfile als Parameter aus.

2.8 remotemanager.sh

Das remotemanager.sh wird vom sshlogin.sh-Skript aufgerufen und übernimmt Wrapperfunktionalitäten und steuert quasi die Kommunikation: Der Ablauf im remotemanager.sh sieht folgendermassen aus:

Das als erster Parameter mitgelieferte Konfigurationsfile (Variable FiletoRun) wird aufgerufen. Damit stehen alle notwendigen variablen zur Verfügung, so z.B. auch die Definitionen zu den Signalisationsdateinamen, Logfilenamen, etc.

Das remotemanager.sh-Skript kann zusätzlich mit einem zweiten Parameter (command) aufgerufen werden. Damit lassen sich interne Prozeduren triggern.

Ohne „command“ wird zuerst die remote_start_method geprüft und die entsprechende Prozedur remotestart_directly oder remote_start_by_cron aufgerufen. Bei remotestart_directly wird das remotemanager-Skript nochmals mit dem command started_directly aufgerufen. Bei remote_start_by_cron wird über die interne Prozedur croncmd ein cron-Job definiert, welcher in der nächsten Minute abgearbeitet werden soll. Als zu startende Datei sind dabei abermals das remotemanager.sh-Skript und das Konfigurationsfile angegeben, neu mit dem command started_by_cron.

> Hinweis:
>
> Der cron-Job ist als minütlich wiederkehrender Job eingetragen, da sich einmalige Jobs nicht per cron nicht definieren lassen. Mittels des atd-Daemons wäre die Definition von einmaligen Jobs möglich. Der atd-Daemon muss aber u.U. auf den Systemen zuerst installiert und/oder aktiviert werden. Um möglichst wenig Abhängigkeiten zu haben, wurde deshalb auf diese Implementierungsvariante verzichtet.

Bei Started_by_cron wird der minütlich wiederkehrende Job wieder gelöscht. Dadurch

Sowohl bei started_by_cron, als auch bei started_directly wird je nach remote_su_method entweder die Prozedur su_method_su oder su_method_expect aufgerufen.

Die Prozedur su_method_su prüft mittels der Prozedur sudo_permission_check, ob sudo-Rechte vorhanden sind. Ist dies der Fall, so kann der eingeloggte Benutzer mittels `sudo -S -l` alle Dateien, welche er mittels sudo aufrufen darf, auflisten. Sollte dies nicht der Fall sein, so werden die notwendigen Berechtigungen mittels der Prozedur sudo_permission_set gesetzt. Danach wird mittels sudo erneut das remotemanager-Skript und das Konfigurationsfile aufgerufen, neu mittels dem command su_done.

Die Prozedur su_method_expect ruft mittels einem expect-Konstrukt erneut das remotemanager-Skript und das Konfigurationsfile auf, neu mittels dem command su_done.

In der Prozedur su_done() wird im Modus DISTRIBUTING geprüft, ob ein Statusflagfile end.tag für die Verarbeitung (Variable Remotedonetagfile) bereits vorhanden ist. Ist dies der Fall, so würde dies bedeuten, dass das Scanlet bereits einmal den Modus Distributing durchlaufen hat, allerdings nicht den Modus Collecting. Sicherheitshalber wird deshalb 30 Sekunden gewartet.

Hintergrundinformationen zur gewählten Implementierung:

> Angenommen ein Scanlet wird zweimal hintereinander aufgerufen und die gewählte Wartezeit zwischen Distributing und Collecting ist zu klein gewählt: Sollte der Modus Collecting vom ersten Durchlauf doch noch erfolgen, so kann die sich abzeichnende Konfliktsituation womöglich noch verhindert werden. Die zu vermeidende Konfliktsituation wäre, dass man im Modus Collecting das temporäre Scanletverzeichnis löscht und parallel dazu durch den bereits gestarteten zweiten Durchlauf nochmals das gleiche Scanlet im Modus Distributing verarbeitet.

Nach der Prüfung der Statusflagfile end.tag wird ein allfällig vorhandenes Logfile (Variable Remotelogfile) und Errorlogfile (Variable Remoteerrorlogfile) gelöscht.

Nach diesen Vorbereitungsschritten wird nun das eigentliche, scanletspezifische Skript scanletjob.sh (Variable remotestartfilename) gestartet.

Im Modus COLLECTING wird in der Prozedur su_done() das Statusflagfile end.tag für die Verarbeitung (Variable Remotedonetagfile) gelöscht. Ist der Wert der Variablen cleanupremotepath auf „Y“ gesetzt, wird das komplette Scanletverzeichnis auf dem Zielsystem gelöscht.

2.9 Scanletjob.sh

Der Scanletentwickler implementiert in diesem Skript die gewünschte Funktionalität.

3 Erstellen eines Scanlets

Beim Erstellen eines eigenen Scanlets ist folgendermassen vorzugehen:

1. Das CoopSSLE-Templatescanlet ist auf den gewünschten Distributionsserver zu kopieren.
2. Für das Hauptverzeichnis CoopSSLE_2.01 ist ein anderer Name zu wählen. Es empfiehlt sich, eine Konvention zu vereinbaren, entweder basierend auf den Scanletinhalt (z.B. NewSBVersion400) oder auf einer Nummerierung, bzw. auf ein Datum (z.B. Patch21032009).
3. Die gewünschten Parameter sind im `scanletcfg.sh` zu definieren.
4. Mittels dem Befehl `chmod a+x scanletstarter.sh` ist sicherzustellen, dass das scanletstarter.sh als bash-Skript ausgeführt werden kann.
   Hinweis: Dies ist für alle bash-Skripts (.sh) auszuführen, sollten die Skripts erstmalig von einem Windows-System hinkopiert worden sein.
5. Im Remotescripts-Verzeichnis ist die Datei `scanletjob.sh` zu editieren.
6. In der Datei `list` sind die gewünschten Systeme einzupflegen. Hierbei können als Clientangaben FQDN oder IP-Adressen definiert werden.
7. In der Datei `recipients` sind die gewünschten Email-Empfänger zu definieren.

Niemals ein ungetestetes, nicht zweitgeprüftes Scanlet auf produktive Systeme massenverteilen!

Getestet bedeutet in der Testumgebung akribisch geprüft!

Niemals ein ungetestetes, nicht zweitgeprüftes Scanlet auf produktive Systeme massenverteilen!

Zweitgeprüft bedeutet die Prüfung durch einen anderen Scanletentwickler!

Niemals ein ungetestetes, nicht zweitgeprüftes Scanlet auf produktive Systeme massenverteilen!

Produktives Verteilen zuerst mittels ausgewählten Piloten, erst danach eine Massenverteilung lancieren!

Das Scanlet muss unter dem root-Account gestartet werden.

Beispiel für den Aufruf:

```
. /tmp/NewSBVersion400/scanletstarter.sh
```