hochschule mannheim

Fakultät für Informatik
Entwicklungsprojekt CEP des Studiengang CSB

Benutzerhandbuch

Abgabetermin: 29.06.2021

Vorgelegt von:

Philip Steinlein - 1921236
Mike Kittelberger - 1921224
Nicolas Biundo - 1924333
Gabriel Kremensas - 1925874
Bianca Knittel - 1921825
Simon Höfer - 1821059

Inhaltsverzeichnis

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

1. Einleitung

Das Benutzerhandbuch zu der Software EZAudit, erläutert im folgenden alle Punkte, um die Software aufsetzen und nutzen zu können. Desweiteren wird erläutert, wie die Software auf den Einsatz vorbereitet und wie Auditschritte an Kundenbedürfnisse angepasst werden können. Zudem wird darauf eingegangen wie das Programm durchgeführt wird, es wird auch darauf eingegangen, wie man die Ergebnisse auswerten kann.

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

2. Zuschneiden der Software auf den Kunden

Im folgenden Abschnitt wird erläutert, wie die Software auf den jeweiligen Kunden angepasst werden kann. Dabei wird auf das Framework, die Auswahl des Betriebssystems, dem Definieren neuer Auditschritte und auf die Verbosity Funktion eingegangen.

2.1 Einrichten des EZ-Audit Frameworks

Das EZ-Audit Framework wird als .exe ausgeliefert, wodurch keine weiteren Einrichtungsschritte, außer dem Download erforderlich sind.

Die config.json wird als Standard gewählt wenn keine andere Config angegeben wurde.
(Dies wäre zum Beispiel über eine Flag möglich)

2.2 Globale Einstellungen in der .json

Um das richtige OS-System auszuwählen muss die ____config.json valide geschrieben werden. Dies bedeutet syntaktisch und semantisch richtig aufgebaut sein.

```
“os” : “linux”,
“verbosity” : 4,
“maxOutputLength” : 500
“Censor” : [“//d”],
“commands” : [ …
```

In der ersten Zeile, kann je nach Anwendungsfall das Betriebssystem angegeben werden:

“os”: “linux”/”windows”(String)

Daraufhin kann die Verbosity angegeben werden, also der Grad der Informationen im Debug.log, dies ist Skalierbar von 1-5. Ohne Angabe dieses Parameters nimmt er den Wert 4 an.

1/5 entspricht: Fatal, loggt fatale Fehler die das Programm beenden.
2/5 entspricht: Error, loggt zudem auch Fehler innerhalb Module.
3/5 entspricht: Warning, loggt zudem auch noch Warnings in dem Code.
4/5 entspricht: Information, loggt zudem auch die einzelnen Ergebnisse und die einzelnen Commands/ Audit Schritte.
5/5 entspricht: Debug, loggt alles was möglich ist, vollständige Ausgabe aller Informationen.

Über die “maxOutputLength” kann eingestellt werden, ab welcher String Länge der Output in dem Debug.log/ Result Report in einer .txt File gespeichert wird. Wenn man nichts angibt, dann ist der Standardwert 350. Man kann auch -1 angeben, es wird dann nicht gekürzt, egal wie lang die Ausgabe ist.
“maxOutputLength” : “Integer” (Integer)

2.3 Ausgabe sanitisieren

Sie können Muster oder ganze Wörter, welche geschwärzt/sanitisiert werden sollen angeben werden, dies kann einerseits global (), als auch auf einzelne Auditschritte möglich angewendet werden. Der passende Befehl hierzu heißt:
“censor” : [“Zu zensierender Teil”, “\\d”] (Array(String))

2.4 Aufbau Auditschritte

```
{
  "os": "linux",
  "verbosity": 4,
  "maxOutputLength": 1000,

  "commands": [
    {
      "name": " Ensure mounting of cramfs filesystems is disabled",
      "steps": [
        {
          "module": "modprobe",
          "parameter": {
            "dryRun": true,
            "version": true,
            "moduleName": "cramfs"
          },
          "comparison": "==",
          "expected": "install /bin/true"
        },{
```

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

Um neue Auditschritte zu definieren, schreiben sie in das command-Array

```
“commands” : [
      {
      ---Ein Auditschritt---
      }
      {
      ---Ein Auditschritt---
      }
      …
]
```

2.3.1 Definieren neuer Auditschritte

Um diese Auditschritte aufzubauen, muss folgendem Muster gefolgt werden alle Modulnamen, Parameters etc sind in camelCase angegeben :

“name”: “Name hier einfügen” (Beachte: camelCase(String))
“steps”: [
Der nun Folgende Step, könnte mehrmals hintereinander eingefügt werden, darauf wird später aber noch einmal explizit eingegangen.
{
“module”: “Gewünschtes Modul hier einfügen”
“(Moduldokumentation startet bei Seite 9)”
“parameter”: {
“Parameter, welche das Modul benötigt, weiter unten in diesem Formular finden sie den Aufbau der Module”
“Beachte, eventuell sind mehrere Parameter pro Modul nötig”
}
“comparison”: “Entweder ==, !=, contains, >=, >, < oder <=. Es wird erkannt ob sich ein Integer oder ein String angegeben wurde.”
“expected”: “Erwarteter Wert, mit dem der Vergleich durchgeführt werden soll, Integers sowie Strings werden in Anführungszeichen geschrieben” (“3”, “True”)

Falls ein Integer expected ist und der Wert ein String ist, wird im Result Report im Description Feld auf diesen Fehler hingewiesen. Über “contains” wird geprüft, ob

gewisse Wörter oder Abfolgen in der Ausgabe sind (“contains”: “die”, wenn nun der Wert “Um diese Auditschritte”, dann wird der Schritt erfolgreich sein).

“onFailure” : {...
}

}

2.3.2 OnFailure und OnSuccess

```
{
  "name": "Check for administrive accounts (with UID 0)",
  "steps": [
    {
      "module": "moduleName",
      "parameter": {
        "argument1": true,
        "argument2": "argument",
        "argument3": 5
      },
      "comparison": "!=",
      "expected": "",

      "onSuccess": {
        "module": "grep",
        "usePipe": true,
        "parameter": {

          "matchingControl": {
            "invertMatch": true,
            "patterns": "'^#|^root:'"
          }
        },
        "comparison": "!=",
        "expected": ""
      }
    }
```

Mit OnSuccess kann in der Config File direkt der nächste Schritt angegeben werden, welcher ausgeführt werden soll, wenn die Comparison davor dem Expected entspricht. Das gleiche gilt auch für OnFailure, es wird nur ausgeführt wenn die Comparison False ausgibt.

```
"onSuccess": {
            "module": "...",
            ... ,
            "parameter": {

              ...
              }
            },

"onFailure: {
            "module": "...",
            ... ,
            "parameter": {

              ...
              }
            },
```

Allgemein gilt jedoch das folgende wenn man eine Config baut:

Die Module sind hierbei der entscheidende Faktor, es dürfen nur Module gewählt werden, die zu dem ausgewähltem OS passen.
(Mit der Ausnahme der Force Flag auf diese wird auf Seite 7 eingegangen)

2.3.3 Flags

--help: (-h) Gibt alle Flags auf der Konsole aus, die benutzt werden können.
--config: (-c) Gibt dem Nutzer*in die Chance den Pfad zur Config anzugeben.
Syntax: --config=PATH / -c=PATH (Ohne Leerzeichen)
--verbosity: (-v) Gibt dem Nutzer*in die Möglichkeit die Verbosity umzustellen, weiteres zu Verbosity siehe Seite 4. Die Verbosity aus der Flag hat eine höhere Priorität als die aus der config.
Syntax: --verbosity=5 / -v=5 (Ohne Leerzeichen)
--noZip: (-z) Ausgabe ist nicht in einer Zip Datei.
--force: (-f) Überspringt die OS-Validierung, bzw den Sanity Check, wenn ein Modul angegeben wird, welches nicht zu dem OS gehört welches erwartet ist, bzw in der Config Json angegeben wurde, versucht das Programm trotzdem die JSON vollständig auszuführen.
--dryRun: (-d) Validiert die Config, ob JSON syntaktisch korrekt ist und ob die Module die korrekten Parameter haben. Überprüft auch ob das Module zu dem OS passen (Wenn Linux Module genutzt muss OS auch als Linux angegeben sein).

Für jede Flag gibt es auch noch eine Shortflag, diese steht in Klammern, hinter der Longflag.

3. Module

3.1 Module in der Config

Alle Steps sind Key-Value-Paare im JSON Format, welche in camelCase geschrieben werden, das heißt, dass die Anfangsbuchstaben grundsätzlich klein geschrieben sind und falls es sich um ein zusammengesetztes Wort handelt wird der erste Buchstabe des zweiten Wortes groß geschrieben.
Von Außen betrachtet sind alle Module gleich aufgebaut und unterscheiden sich lediglich in ihren Parametern. In den folgenden Tabelle ist angegeben welche Parameter jedes Modul benötigt, um welchen Datentyp es sich handelt und an einem kleinen Beispiel wird jeweils der Aufbau veranschaulicht. Sollte ein Modul keine Parameter enthalten, muss bei der Modul Erstellung dennoch “parameter” : {} mit angegeben werden (s.u.).
Bei jedes Modul kann um einen Vergleichsoperator ergänzt werden, sowie ob dieses Modul in der Pipe gespeichert werden soll und wird ein bestimmter Wert erwartet kann dieser ebenfalls definiert werden:

“module” : “modulName”,
“parameter” : {
…
}
“comparison” : “==”
“usePipe” : true [boolean],
“Allow Failure” : false [boolean],
“needsElevation” : true [boolean],
“expected” : “”

Ein Step wird solange in der Pipe gespeichert, bis ein Step folgt bei welchem usePipe nicht gesetzt wurde.
Wird des Weiteren der Parameter Admin auf true gesetzt, wird den Parametern sudo vorangestellt.
Des Weiteren sollte ein Modul zwingend Adminrechte erfordern, kann dies durch Setzen des Parameters needsElevation verwirklicht werden. Dies wäre zum Beispiel auf Shadow der Fall, wenn das Modul grep ausgeführt werden soll.
Weiterführend können Rückgaben der Module mit comparison und expected überprüft werden, ob die Rückgabe dem erwarteten Wert entspricht.

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

Mit allowFailure können Module einen Fehler erlauben, so dass wenn ein Fehler eintritt nicht der Step als gefailed betrachtet wird. Ausnahme hiervon sind Errors, tritt einer ein wird der Step trotz allowFailure = true, der Step als gefailed betrachtet.
Um sensible Informationen zu schützen, können diese sanitisiert werden. Dies erfolgt durch censor, welcher ein Array aus Strings erhält, die Regex Ausdrücke repräsentieren und verhindert das sensible Informationen angezeigt werden.
Zudem kann in jedem Modul dontSaveArtefact mitangegeben werden, sollte dieses nicht als Artefakt gespeichert werden.

3.1.1 Linux Module

3.1.1.1 Alias

Module	alias
Optionen	-
Pfad	-
Beschreibung	Kann Befehle ersetzen, um Zeit und Tippaufwand zu ersparen.
Beispiel Aufbau	
“module” : “alias”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.2 Apparmor_Status

Module	apparmor_status
Optionen	-
Pfad	-
Beschreibung	Mit apparmor_status wird überprüft, ob der Prozess autorisiert ist den gewünschten Vorgang auszuführen.
Beispiel Aufbau	
“module” : “apparmor_status”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.3 Apt

Module	apt
Optionen	cache [boolean] upgrade [boolean] purge [boolean] keylist [boolean] packageName [string]
Pfad	-
Beschreibung	Apt dient der Paketverwaltung und unterstützt bei der Suche, Installation und Aktualisierung von Programmpaketen.
Beispiel Aufbau	
“module” : “apt”, “parameter” : { “cache” : true, “upgrade” : false }, “comparison” : “!=”, “expected” : ””	

3.1.1.4 Auditctl

Module	auditctl
Optionen	-
Pfad	-
Beschreibung	Mit auditctl können Regeln für das Logging definiert werden.
Beispiel Aufbau	
“module” : “auditctl”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

3.1.1.5 AuthselectCurrent

Module	authselectCurrent
Optionen	-
Pfad	-
Beschreibung	AuthselectCurrent konfiguriert Authentifizierungen und Identitäten bei der Auswahl von diversen Profilen.
Beispiel Aufbau	
“module” : “authselectCurrent”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.6 Awk

Module	awk
Optionen	-F → fieldSeparator [boolean]
Pfad	Datei
Beschreibung	Awk dient dem zeilenweise editieren und analysieren von Texten.
Beispiel Aufbau	
“module” : “awk” “parameter” : { “fieldSeparator” : true, }, “comparison” : “!=”, “expected” : ””	

3.1.1.7 Cat

Module	cat
Optionen	path [string]
Pfad	Datei
Beschreibung	Cat kommt zum Einsatz bei Verkettungen von Dateien und kann zusätzlich auch Dateiinhalte im Terminal anzeigen.
Beispiel Aufbau	
“module” : “cat”, “parameter” : { “path” : “/home/user/goPath/Just-Go-IT/Audit-Framework/configFiles/configCat.json” , }, “comparison” : “!=”, “expected” : ””	

3.1.1.8 Certuil

Module	certutil
Optionen	-
Pfad	-
Beschreibung	Mit certutil können zertifizierte Datenbanken erzeugt oder aufgerufen werden.
Beispiel Aufbau	
“module” : “certuil”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.9 Command

Module	command
Optionen	-v → printDescription [boolean] -V → verboseDescription [boolean] target [string]
Pfad	-
Beschreibung	Zeigt Informationen zu den einzelnen Modulen an.
Beispiel Aufbau	
“module” : “command”, “parameter” : { “printDescription” : true , }, “comparison” : “!=”, “expected” : ””	

3.1.1.10 Crontab

Module	crontab
Optionen	admin [boolean]
Pfad	-
Beschreibung	Mit crontab wird cron gesteuert. Zudem verwaltet crontab automatsierte Sicherungen, synchronisiert Dateien zwischen Remotecomputern und löscht temporäre Ordner und dient im allgemeinen der Systemadministration.
Beispiel Aufbau	
“module” : “crontab”, “parameter” : { “admin” : true , }, “comparison” : “!=”, “expected” : ””	

3.1.1.11 Cut

Module	cut
Optionen	-c → characters [boolean] -d → delimiter [boolean] target [string]
Pfad	-
Beschreibung	Cut extrahiert Teile aus Dateien.
Beispiel Aufbau	
“module” : “cut”, “parameter” : { “delimeter” : true , }, “comparison” : “!=”, “expected” : ””	

3.1.1.12 Df

Module	df
Optionen	-l → local [boolean]
Pfad	Datei
Beschreibung	Zur Überwachung der gesamten Dateisysteme wird df verwendet.
Beispiel Aufbau	
“module” : “df”, “parameter” : { “local” : true , }, “comparison” : “!=”, “expected” : ””	

3.1.1.13 Dnf

Module	dnf
Optionen	repoList [boolean] checkUpdate [boolean]
Pfad	-
Beschreibung	Dies ist ein Paketmanagement-System speziell für RPM-Softwaresysteme.
Beispiel Aufbau	
“module” : “dnf”, “parameter” : { “checkUpdate” : true , “repoList” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.14 Dpkg

Module	dpkg
Optionen	-s → search [boolean] -S → status [boolean] -l → listPattern [string] --verify → verify [boolean]
Pfad	-
Beschreibung	Dpkg dient der Paketverwaltung und unterstützt bei der Suche, Installation und Aktualisierung von Programmpaketen.
Beispiel Aufbau	
“module” : “dpkg”, “parameter” : { “search” : true , “verify” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.15 Echo

Module	echo
Optionen	-
Pfad	-
Beschreibung	Echo ermöglicht eine Kommandozeilen Ausgabe.
Beispiel Aufbau	
“module” : “echo”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.16 Find

Module	find
Optionen	target [string]
Pfad	Verzeichnis
Beschreibung	Hiermit wird die Verzeichnis-Hierarchie rekursiv nach einem bestimmten Kriterium durchsucht.
Beispiel Aufbau	
“module” : “find”, “parameter” : { “target” : "/ -nouser -o -nogroup 2>/dev/null", }, “comparison” : “!=”, “expected” : ””	

3.1.1.17 Firewall

Module	firewall
Optionen	--state → state [boolean] -get-active zone → getActiveZone [boolean] -get-default zone → getDefaultZone [boolean]
Pfad	-
Beschreibung	Das ist der Client für den Firewall Daemon.
Beispiel Aufbau	
“module” : “firewall”, “parameter” : { “state” : true, “getActiveZone” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.18 Grep

Module	grep
Optionen	target [string] searchPattern [string] admin [boolean] patternSyntax [struct] -G → basicRegexp [boolean] -E → extendRegexp [boolean] -F → fixedStrings [boolean] -P → perlRegexp [boolean] -x → lineRegexp [boolean] matchingControl [struct] -e → patterns [string] -f → patternFile [string] -i → ignoreCase [boolean] --no-ignore-case → noIgnoreCase [boolean] -v → invertMatch [boolean] -w → wordRegexp [boolean] outputControl [struct] -c → count [boolean] -L → filesWithoutMatches [boolean] -l → filesWithMatches [boolean] -m → maxCount [int] -o → onlyMatching [boolean] -r → recursive [boolean]
Pfad	Dateipfad
Beschreibung	Mit grep können Dateien nach bestimmten Textmustern durchsucht werden.
Beispiel Aufbau	

“module” : “grep”,
“parameter” : {
“file”: “/etc/ssh/ssh_config”,
“admin” : true,
“matchingControl” :{ → Verschachtelungen der Operationen s.o.
“ignoreCase” : true,
“patterns” : “'^\s*Defaults\s+([^#]+,\s*)?use_pty(,\s+\S+\s*)*(\s+#.*)?$'”
}
“target” : “/etc/sudoers /etc/sudoers.d/*”	

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

}, “comparison” : “!=”, “expected” : ””

3.1.1.19 Grpck

Module	grpck
Optionen	admin [boolean] -r → readOnlyMode [boolean] -q → optionQ [boolean] group [string] shadow [string]
Pfad	-
Beschreibung	Mit grpck verifiziert Einträge auf das richtige Format und und Gültigkeit.
Beispiel Aufbau	
“module” : “grpck”, “parameter” : { “admin” : true, “optionQ” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.20 Id

Module	id
Optionen	-
Pfad	-
Beschreibung	Id ermittelt user und Gruppennamen.
Beispiel Aufbau	
“module” : “id”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.21 Ip

Module	ip
Optionen	showAddress [boolean] routeList [boolean] old [string]
Pfad	-
Beschreibung	Mit Ip können Konfigurationsinformationen der Netzwerke überprüft werden.
Beispiel Aufbau	
“module” : “ip”, “parameter” : { "old": "/sbin/ifconfig -a" }, “comparison” : “!=”, “expected” : ””	

3.1.1.21 Iptables

Module	iptables
Optionen	options [string] -v → verbose [boolean] -n → nummeric [boolean]
Pfad	-
Beschreibung	Iptables dient als Paketfilter des Kernels und kontrolliert welche Pakete passieren dürfen.
Beispiel Aufbau	
“module” : “iptables”, “parameter” : { “verbose” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.22 Ip6tables

Module	ip6tables
Optionen	options [string] -v → verbose [boolean] -n → nummeric [boolean]
Pfad	-
Beschreibung	Siehe auch iptables. Hier wird jedoch ipv6 statt ipv4 abgedeckt.
Beispiel Aufbau	
“module” : “ip6tables”, “parameter” : { “nummeric” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.23 Journalctl

Module	journalctl
Optionen	admin [boolean]
Pfad	-
Beschreibung	Um auf Daten, die im Journal gespeichert sind, zugreifen oder verändern zu können benötigt man den Befehl journalctl.
Beispiel Aufbau	
“module” : “journalctl”, “parameter” : { “admin” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.24 Lastlog

Module	lastlog
Optionen	-
Pfad	-
Beschreibung	Lastlog dokumentiert die letzten Logins jedes Benutzers.
Beispiel Aufbau	
“module” : “lastlog”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.25 Ls

Module	ls
Optionen	path [string]
Pfad	Verzeichnis
Beschreibung	ls zeigt den Inhalt eines Verzeichnisses oder Ordners an.
Beispiel Aufbau	
“module” : “ls”, “parameter” : { "path": "/etc/sudoers" }, “comparison” : “!=”, “expected” : ””	

3.1.1.26 Lsb_release

Module	lsb_release
Optionen	-
Pfad	-
Beschreibung	Lsb_release gibt Auskunft über die gesamte Linux Distribution.
Beispiel Aufbau	
“module” : “ls”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.27 Lsmod

Module	lsmod
Optionen	-
Pfad	-
Beschreibung	Mit lsmod wird der Status der Module angezeigt, die sich im Kernel befinden.
Beispiel Aufbau	
“module” : “ls”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.28 Modprobe

Module	modprobe
Optionen	-n → dryRun [boolean] -V → verbose [boolean] -v → version [boolean] -a → all [boolean] Zudem mit grep kombinierbar.
Pfad	Modulname
Beschreibung	Um ein Modul zur Laufzeit in den Kernel zu Laden, verwendet man modprobe und übergibt ihm den Namen des Moduls. Modprobe lädt daraufhin die Datei mit möglichen Abhängigkeiten.
Beispiel Aufbau	
“module” : “modprobe”, “parameter” : { "all" : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.29 Mount

Module	mount
Optionen	admin [boolean] -t → type [string] -o → options [string]
Pfad	Einhängepunkt
Beschreibung	Mit mount können Dateien manuell an diversen Punkten eingebunden werden. Wichtig ist dabei den Dateisystemtyp, da zu mountende Gerät und den Mountpoint mit anzugeben.
Beispiel Aufbau	
“module” : “mount”, “parameter” : { "admin" : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.30 Nft

Module	nft
Optionen	-
Pfad	-
Beschreibung	Mit nft können alle Regeln für Pakete in Netzwerken, die auf dem System laufen festgelegt werden, wie zum Beispiel die Weiterleitung, Modifikation und Ablehnung von Paketen.
Beispiel Aufbau	
“module” : “nft”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

3.1.1.31 Nmcil

Module	nmcli
Optionen	-t → terse [boolean] radioAll [boolean]
Pfad	-
Beschreibung	Die Aufgabe von nmcli ist die Überwachung der Netzverbindung und bei Abbrüchen diese wiederherzustellen.
Beispiel Aufbau	
“module” : “nmcli”, “parameter” : { "terse" : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.32 Ntpq

Module	ntpq
Optionen	-
Pfad	-
Beschreibung	Gleicht Systemzeit mit einem zentralen Zeitserver ab.
Beispiel Aufbau	
“module” : “ntpq”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.33 Ps

Module	ps
Optionen	-eZ → addSecurityData [boolean] -ef → standardSyntax [boolean] aux [boolean] target [string]
Pfad	-
Beschreibung	Ps listet alle Prozesse mit ihrem Status auf.
Beispiel Aufbau	
“module” : “ps”, “parameter” : { "addSecurityData" : true, “aux” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.34 Rpcinfo

Module	rpcinfo
Optionen	-
Pfad	-
Beschreibung	Rpcinfo gibt wie der Name schon vermuten lässt alle Informationen bezüglich des RPC (Remote Procedure Call) aus.
Beispiel Aufbau	
“module” : “ps”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.35 Rpm

Module	rpm
Optionen	target [string] -q → query [boolean] -pq → queryAll [boolean] -Va → verify [boolean]
Pfad	Paketname
Beschreibung	Rpm ist ein Paketmanager.
Beispiel Aufbau	
“module” : “ps”, “parameter” : { “verify” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.36 Sestatus

Module	sestatus
Optionen	-v → optionV [boolean]
Pfad	-
Beschreibung	Mit sestatus wird angegeben ob der Status für SELinux enabled oder disabled ist und ob die Richtlinien enforced oder permissive sind.
Beispiel Aufbau	
“module” : “sestatus”, “parameter” : { “optionV” : true }, “comparison” : “!=”, “expected” : ””	

3.1.1.37 Set

Module	set
Optionen	-
Pfad	-
Beschreibung	Set zeigt alle Umgebungsvariablen an.
Beispiel Aufbau	
“module” : “set”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.38 Ss

Module	ss
Optionen	argument [string]
Pfad	-
Beschreibung	Um Netzwerkstatistiken aus dem Kernel Space auszulesen verwendet man ss.
Beispiel Aufbau	
“module” : “ss”, “parameter” : { “argument” : “-nptua” }, “comparison” : “!=”, “expected” : ””	

3.1.1.39 Sshd

Module	sshd
Optionen	-
Pfad	-
Beschreibung	Sshd sorgt für eine authentifizierte, und verschlüsselte Datenübertragung, bei der die Datenintegrität garantiert wird.
Beispiel Aufbau	
“module” : “sshd”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.40 Stat

Module	stat
Optionen	-
Pfad	Dateiname
Beschreibung	Stat zeigt Zugriffs- und Änderungs-Zeitstempel von Dateien und Ordnern an. Des Weiteren gibt stat Details zu Rechten, Benutzer und Gruppen der Datei aus und mit den definierten Formaten können die Ausgaben den Wünschen angepasst werden.
Beispiel Aufbau	
“module” : “stat”, “parameter” : { “path” : “/etc/issue” }, “comparison” : “!=”, “expected” : ””	

3.1.1.41 Subscription-manager identity

Module	subscription-manager identity
Optionen	-
Pfad	-
Beschreibung	Listet die Identität mit der ID, einem Sicherheitszertifikat oder der Organisation auf.
Beispiel Aufbau	
“module” : “subscriptionManager”, “parameter”:{ }, “comparison” : “!=”, “expected” : ””	

3.1.1.42 Sysctl

Module	sysctl
Optionen	target [string]
Pfad	Dateiname
Beschreibung	Mit sysctl werden Kernelparameter während der Laufzeit konfiguriert.
Beispiel Aufbau	
“module” : “sysctl”, “parameter” : { "searchPattern" : "kernel\\.randomize_va_space" “target” : “/etc/sysctl.conf /etc/sxsctl.d/*” }, “comparison” : “==”, “expected” : ”kernel\\.randomize_va_space = 2”	

3.1.1.43 Systemctl

Module	systemctl
Optionen	--now → now [boolean] disable → diableUnit [string] is-enabled → isEnabled [string] status [string]
Pfad	-
Beschreibung	Um das Init-System steuern zu können, wird Systemctl als Verwaltungswerkzeug eingesetzt. Dabei werden Dienste verwaltet, der Status überprüft, Änderungen von Systemzuständen und alle bezüglich dem Umgang mit den Konfigurationsdateien behandelt.
Beispiel Aufbau	
“module” : “systemctl”, “parameter” : { "isEnabled” : “tmp.mount” }, “comparison” : “==”, “expected” : ”enabled”	

3.1.1.44 Touch

Module	touch
Optionen	path [string]
Pfad	Datei
Beschreibung	DIe Hauptaufgabe von touch ist es Zugriffs- und Änderungs-Zeitstempel von Dateien zu verändern, gelegentlich wird touch aber auch zur Erzeugung von leeren Dateien verwendet.
Beispiel Aufbau	
“module” : “touch”, “parameter” : { "path” : “/etc” }, “comparison” : “==”, “expected” : ”enabled”	

3.1.1.45 Useradd

Module	useradd
Optionen	-
Pfad	Homeverzeichnis
Beschreibung	Hiermit wird ein neuer User-Account angelegt.
Beispiel Aufbau	
“module” : “useradd”, “parameter”:{ }, “comparison” : “==”, “expected” : ”enabled”	

3.1.1.46 Whereis

Module	whereis
Optionen	target [string]
Pfad	-
Beschreibung	Mit wheris wird nach binären Dateien, Quelldateien und Manual-Seite-Dateien gesucht.
Beispiel Aufbau	
“module” : “whereis”, “parameter” : { "target” : lsof” }, “comparison” : “contains”, “expected” : ”lsof”	

3.1.1.47 Xargs

Module	xargs
Optionen	-0 → null [boolean] arguments [string]
Pfad	-
Beschreibung	Viele Commands ist es möglich auf Dateien oder Dateilisten angewendet zu werden, sollte dies nicht funktionieren kann alternativ xargs verwendet werden.
Beispiel Aufbau	
“module” : “xargs”, “parameter” : { "arguments” : “ls-ld” }, “comparison” : “==”, “expected” : ””	

3.1.1.48 Yum

Module	yum
Optionen	list-security → listSecurity [boolean] list installed → listInstalled [boolean]
Pfad	-
Beschreibung	Yum gibt alle installierten Pakete aus.
Beispiel Aufbau	
“module” : “yum”, “parameter” : { "listInstalled” : “true” }, “comparison” : “==”, “expected” : ”enabled”	

3.1.2 Windows Module

3.1.2.1 Auditpol

Module	auditpol
Optionen	match machineName [string] policyTarget [string] subCategory [string] subCategoryGUID [string] inclusionSetting [string] exclusionSetting [string]
Pfad	-
Beschreibung	Mit auditpol können Keys mit bestimmten Suchmuster ineinander verschachtelt werden, wobei die Ausgabe allen angegebenen Suchmustern entsprechen muss. Finden sich mehrere passende Keys, werden diese in Artefacts gespeichert.
Beispiel Aufbau	

```
“module” : “auditpol”,
“parameter” : {
   "match” : {
          "machineName" :  "T490S",
          "policyTarget" :  "System",
          "subCategory" :  "IPsecDriver",
          "subCategoryGUID" : "{0CCE9213-69AE-11D9-BED3-505054503030}",
         "inclusionSetting" : "No Auditing"
          }
      },
  }
“comparison” : “==”,
“expected” : ””
```

3.1.2.2 Dir

Module	dir
Optionen	path [string]
Pfad	Benötigt Pfad zum Verzeichnis.
Beschreibung	Gibt eine Liste von Dateien und Unterverzeichnissen eine Verzeichnisses aus.
Beispiel Aufbau	
“module” : “dir”, “parameter” : { “path” : “” }, “comparison” : “==”, “expected” : ””	

3.1.2.3 GetContent

Module	getContent
Optionen	path [string]
Pfad	Benötigt Pfad zum Verzeichnis.
Beschreibung	Gibt Infos aus einer Datei im aktuellen Verzeichnis aus.
Beispiel Aufbau	
“module” : “getContent”, “parameter” : { “path” : “” }, “comparison” : “==”, “expected” : ””	

3.1.2.4 GetGpo

Module	getGpo
Optionen	-Guid → guid [string] -Name → name [string] -Domain → domain [string] -Server → server [string] -All → all [boolean]
Pfad	-
Beschreibung	Man erhält alle Informationen zur Group Policy.
Beispiel Aufbau	
“module” : “getGpo”, “parameter” : { “all” : true }, “comparison” : “==”, “expected” : ””	

3.1.2.5 GetItemProperty

Module	getItemProperty
Optionen	-Path → path [string] -LiteralPath → litheralPath [string] -Name → name [string] -Filter → filter [string] -Include → include [string] -Exclude → exclude [string]
Pfad	-
Beschreibung	Mit getItemProperty erhält man Informationen zu Dateien, Verzeichnissen, als auch Registries.
Beispiel Aufbau	
“module” : “getItemProperty”, “parameter” : { “path” : “C:\\Windows” }, “comparison” : “==”, “expected” : ””	

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

3.1.2.6 GetItemPropertyValue

Module	getItemPropertyValue
Optionen	-Path → path [string] -LiteralPath → litheralPath [string] -Name → name [string] -Filter → filter [string] -Include → include [string] -Exclude → exclude [string]
Pfad	-
Beschreibung	Man erhält alle Eigenschaften von einer Datei.
Beispiel Aufbau	
“module” : “getItemPropertyValue”, “parameter” : { "path": "HKLM:\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System", "name": "DisableCAD" }, “comparison": "==", "expected": "1"	

3.1.2.7 GetProcess

Module	getProcess
Optionen	-Name → name [string]
Pfad	-
Beschreibung	Gibt alle Prozesse aus die aktuelle auf dem Pc laufen.
Beispiel Aufbau	
“module” : “getProcess”, “parameter” : { }, “comparison": "==", "expected": ""	

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

3.1.2.8 IsInstalled

Module	isInstalled
Optionen	-Name → name [string]
Pfad	-
Beschreibung	Überprüft ob eine Datei/ Registry installiert ist.
Beispiel Aufbau	
“module” : “isInstalled”, “parameter” : { “name” : “skype” }, “comparison": "!=", "expected": ""	

3.1.2.9 Net

Module	net
Optionen	accounts [boolean] net view → view [boolean] net group → group [boolean] net localGroup → localGroup [boolean] net start → start [boolean] net share → share [boolean] net user → user [boolean]
Pfad	-
Beschreibung	Net dient der Analyse des Netzwerkes eines Computers.
Beispiel Aufbau	
“module” : “isInstalled”, “parameter” : { “view” : true, “user” : true }, “comparison": "!=", "expected": ""	

3.1.2.10 Secedit

Module	secedit
Optionen	pattern [string]
Pfad	-
Beschreibung	Secedit vergleicht die aktuellen Sicherheitskonfiguration mit vorgegebenen Sicherheits Vorlagen und analysiert auf diese Weise die Systemsicherheit.
Beispiel Aufbau	
“module” : “secedit”, “parameter” : { “pattern” : “PasswordHistorySize” }, “comparison": "==", "expected": "24"	

3.1.2.11 SelectString

Module	selectString
Optionen	-Path → path [string] -quiet → quiet [boolean] -caseSensitive → caseSensitive [boolean] -Pattern → pattern [string]
Pfad	-
Beschreibung	Findet Textausschnitte in Dateien.
Beispiel Aufbau	
“module” : “selectString”, “parameter” : { “pattern” : “wlan” }, “comparison": "!=", "expected": ""	

3.1.2.12 WindowsServices

Module	windowsServices
Optionen	name [string] status [string]
Pfad	-
Beschreibung	Sammelt alle Windows Funktionalitäten.
Beispiel Aufbau	
“module” : “windowsServices”, “parameter” : { “name” : “vss” }, “comparison”: "!=", "expected": ""	

3.1.3 Module für Linux und Windows

3.1.3.1 Bash

Module	bash
Optionen	script [string] path [string]
Pfad	-
Beschreibung	Bash führt Befehle aus, die aus einer Standardeingabe oder Dateien eingelesen werden.
Beispiel Aufbau	
“module” : “bash”, “parameter” : { }, “comparison”: "!=", "expected": ""	

3.1.3.2 Broken

Module	broken
Optionen	-
Pfad	-
Beschreibung	Wird im Parser ein neues Modul erzeugt und der new Aufruf schlägt fehl, wird anstelle einer Nullpointerexception im Executer, die Info hinterlegt, weswegen das Modul nicht erzeugt werden kann.
Beispiel Aufbau	
“module” : “broken”, “parameter” : { }, “comparison”: "!=", "expected": ""	

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

3.1.3.3 Ecma

Module	ecma
Optionen	script [string] path [string]
Pfad	-
Beschreibung	Grundsätzlich lässt sich in script und path alles einbetten das in ECMA5 verfasst wurde. Implemtierung: - 1. Parameter gibt an mit welchem Modul gearbeitet werden soll - 2. Parameter listet die verwendeten Parameter auf von String zu Interface - 3. Parameter legt fest, ob das Artefakt gesammelt werden soll - 4. Parameter gibt an ob mit Piping gearbeitet werden soll Das Ergebnis ist ein String, welcher das Ergebnis des verwendeten Moduls ausgibt. Die result Funktion gibt eine ECMA Ausgabe zurück, sodass in der Config mit comparison und expected das Ergebnis ausgewertet werden kann.

Beispiel Aufbau

```
“module” : “ecma”,
“parameter” : {
      “path” : “”,
      “script” : “newDate()”
    },
“comparison": "==",
"expected": "’ ’"
```

Muster zur Implementierung

```
executeModule (modulName string, parameter map [string]interface{} saveArtifact bool, usePipe bool, step globalStep) string{
    result, func(result string){
          output = result
    }
}
```

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

3.2 Neue Module erstellen

Um ein neues Modul zu erstellen muss ein struct erstellt werden, welches das Module interface implementiert. Dazu muss es die Methoden new und execute implementieren. Als erstes wird das struct kreiert und alle nötigen Parameter mit ihren Datentypen angegeben:

```
type ModuleTemplate struct {
	argument1 bool
	argument2 string
	argument3 int
}
```

Weiterführend wird in der Funktion New der Namen des Moduls sowie eine Map aus Strings zu Interface übergeben. Wird in dieser Funktion ein Fehler geworfen wird ein Error zurückgegeben ansonsten das Modul.

```
func (w ModuleTemplate) New(p map[string]interface{}) (global.Module, error)
```

Zu Beginn muss definiert werden welche Parameter zwingend erforderlich sind:

```
w.argument2, ok = p["needed"].(string)
	if !ok {
		return nil, errors.New("the key \"needed\" must be set
		and the value must be a \"type\"")
	}
```

Anschließend werden alle Parameter die wahlweise mit angegeben werden können implementiert. Hierfür wird mit switch-cases gearbeitet:

```
for key := range p {
	switch key {
	case "path":
		w.argument1, ok = p["argument1"].(bool)
		if !ok {
			return nil, errors.New("the key \"argument1\" must
			be set and the value must be a \"bool\"")
		}
	default:
		if key != "argument1" {
```

```
                        return nil, errors.New("there is no key called: \""
                        + key + "\" in the module")
                  }
            }
      }
      return &w, nil
```

Abschließend wird in der Funktion Execute der Command dann gebildet. Es handelt sich dabei um eine simple String Verkettung. Mit Hilfe von If-Abfragen kann ermittelt werden, ob in diesem Fall ein bestimmter Parameter verwendet wird und ist dies der Fall wird der Command um diesen erweitert.

```
func (w *ModuleTemplate) Execute(s *global.Step) (output string, err
      error) {
            cmd := ""

            if w.argument1 {
                  cmd += " "
            }

            output, err = interact.ShellPipe(cmd, s)
            if err != nil {
                  return
            }

            artifact.SaveString(output, *s)

            return
}
```

Abschließend wird der Output des Commands im Artefakten Ordner gespeichert.

4. Durchführung des Programms

Um das Programm ausführen zu können müssen einige Schritte noch durchgeführt werden:

Programm Set Up:

Downloaden Sie die aktuelle Version des Programms und entpacken Sie dieses. Sie haben nun eine .exe mit der sie das Programm starten können.

Cross Kompilieren:

Hierfür muss der Source Code heruntergeladen werden, es reicht nicht aus wenn man nur über die .exe verfügt.
Durch setzen der Go Umgebungsvariablen wird Cross Compiling ermöglicht:

“`set GOOS=your OS`”
“`set GOARCH=your CPU architecture`”

Sie können die aktuellen Einstellungen mit dem Befehl
“`go env GOARCH GOOS`” einsehen, um die vorher gesetzten Einträge zu überprüfen.

Sie können eine Liste der möglichen Variablen auch mit dem Befehl
“`go tool dist list`” einsehen.

Wenn Sie nun im Terminal zu dem EZAudit Ordner navigieren, können Sie mit “go build” die .exe erzeugen, welche für das Ziel Betriebssystem passend ist.

5. Auswertung der Ergebnisse

Nach erfolgreicher Durchführung, wird eine .json Datei als ResultReport ausgegeben und samt Logs, gesammelten Artefakten sowie der benutzten Konfigurationsdatei in einem Ordner oder wahlweise als Zip-File abgelegt.

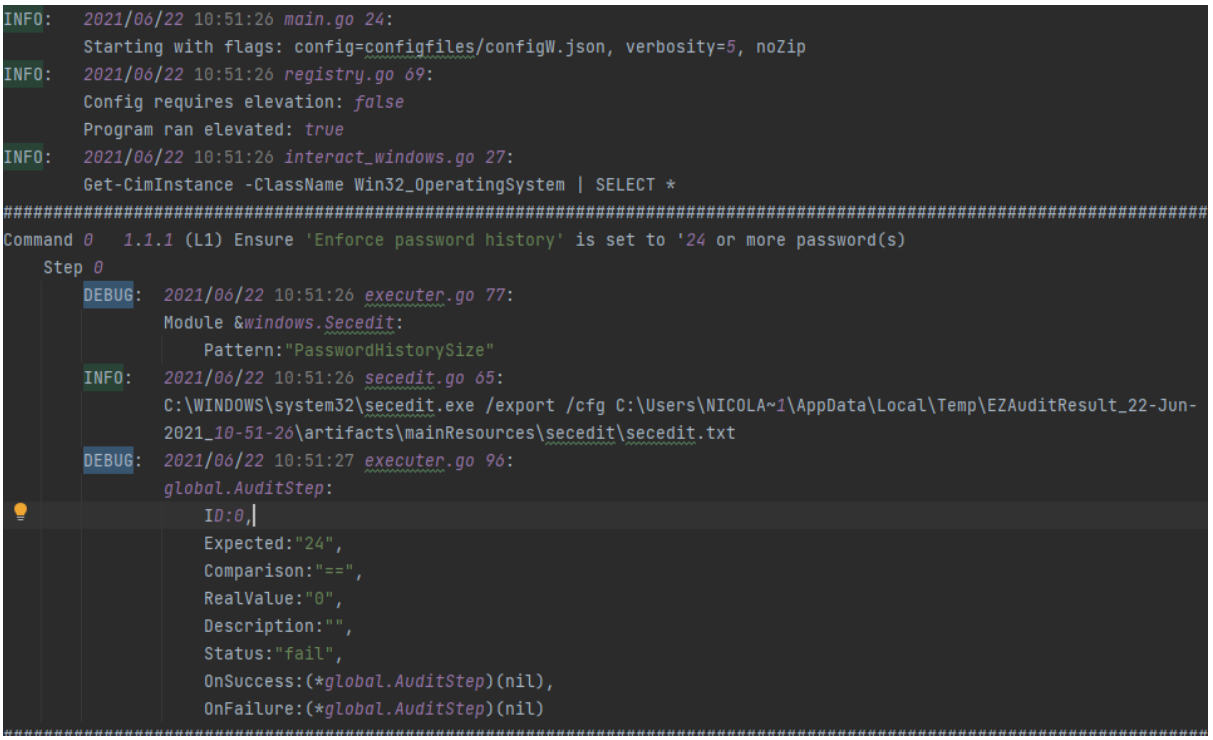
5.1 ResultReport.json


```
{
 "auditSummary": {
  "passedPercentage": "18.87%",
  "passed": 10,
  "failed": 25,
  "errors": 18
 },
 "general": {
  "date": "2021-06-22 10:51:26.4263798 +0200 CEST m=+0.040332401",
  "executionTime": "Elapsed time 1.97s",
  "interact": "Windows operating system | Product Name: Microsoft Windows 10 Pro | Current Build Version: 19042 | Versio
  "admin": true,
  "user": " (ID: S-1-5-21-2068594027-868955951-3733561519-1001)",
  "userName": "NB-WIN10\\Nicolas Biundo"
 },
 "audit": [
  {
   "name": "1.1.1 (L1) Ensure 'Enforce password history' is set to '24 or more password(s)",
   "auditID": 0,
   "steps": [
    {
     "auditStepID": 0,
     "expected": "24",
     "comparison": "==",
     "realValue": "0",
     "status": "fail"
    }
   ],
   "status": "fail"
  },
```


In dem ResultReport wird ein grundlegender Überblick über die abgeschlossenen Audit Schritte dargestellt, sowie auch die “general Informations” wie Time Stamp und spezifische Systeminformationen gelistet.
Des weiteren wird daraus sofort ersichtlich welche Audit Schritte als “passed” oder “failure” eingestuft werden.

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

5.2 Debug.log



```
INFO:   2021/06/22 10:51:26 main.go 24:
        Starting with flags: config=configfiles/configW.json, verbosity=5, noZip
INFO:   2021/06/22 10:51:26 registry.go 69:
        Config requires elevation: false
        Program ran elevated: true
INFO:   2021/06/22 10:51:26 interact_windows.go 27:
        Get-CimInstance -ClassName Win32_OperatingSystem | SELECT *
#########################################################################################################################
Command 0   1.1.1 (L1) Ensure 'Enforce password history' is set to '24 or more password(s)
    Step 0
        DEBUG:  2021/06/22 10:51:26 executer.go 77:
                Module &windows.Secedit:
                    Pattern:"PasswordHistorySize"
        INFO:   2021/06/22 10:51:26 secedit.go 65:
                C:\WINDOWS\system32\secedit.exe /export /cfg C:\Users\NICOLA~1\AppData\Local\Temp\EZAuditResult_22-Jun-
                2021_10-51-26\artifacts\mainResources\secedit\secedit.txt
        DEBUG:  2021/06/22 10:51:27 executer.go 96:
                global.AuditStep:
                    ID:0,
                    Expected:"24",
                    Comparison:"==",
                    RealValue:"0",
                    Description:"",
                    Status:"fail",
                    OnSuccess:(*global.AuditStep)(nil),
                    OnFailure:(*global.AuditStep)(nil)
#########################################################################################################################
```

Dieser Debug Log wurde mit der Verbosity 5 erstellt, dies ist die detaillierteste Version des Debug Logs welche erreicht werden kann.

Hier kann nachvollzogen werden, warum Audit steps fehlgeschlagen sind oder erfolgreich waren.

```
Expected:"24",
Comparison:"==",
RealValue:"0",
Description:"",
Status:"fail",
```

Im Unterschied zum resultReport.json, werden hier auch jegliche Pfade aufgezeichnet.

Im Header werden alle Fatals, Errors, Warnings und Informationen ausgegeben, im Footer, welcher daraufhin folgt, wird auf jeden Audit Schritt einzeln eingegangen und Dinge wie Expected und RealValue ausgegeben.

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT

5.3 ArtifactsFolder

In der Artifakt Folder Struktur werden alle Artefakte abgelegt, welche in den Modulen gesammlet werden, dies sind einerseits das Ergebnis der Module, andererseits wenn das Modul eine Datei benutzt, oder ein Befehl auf eine Datei ausführt dann wird diese Datei auch abgelegt.

Ebenso, wenn eine Ausgabe zu lang für den Resultreport.json oder Debug.log zu lang ist (maxOutputLength) überschritten siehe …

Hier kann man alle Ausgaben einsehen die zu groß für den Result Report oder Debug.log waren, sie sind alphabetisch sortiert.
Ab welcher Länge der Ausgabe sie als Artifact gespeichert werden hängt von dem Parameter “maxOutputLength” in der Config ab.

6. Go Docs

Zusätzlich zu diesem Benutzerhandbuch haben wir noch eine Go Docs Dokumentation angelegt, welche sie entweder selber generieren können oder über das Mitgelieferte HTML Dokument anschauen können.

Wir haben über GitHub die Go Dokumentation auch gehostet, sie können mit diesem Link also auch darauf zugreifen: https://mike-ki.github.io/EZAuditGoDocs/

Wenn sie es selbst erzeugen wollen:

Benötigen sie dafür das Programm Go Docs und dann können sie in Go Docs “-http:/:8080” in dem Ordner in dem das Programm liegt eingeben oder sie müssen den Path angeben.

Die Offizielle Dokumentation finden sie hier:

https://pkg.go.dev/golang.org/x/tools/cmd/godoc

Letzte Änderung:	29.06.21
Letzter Bearbeiter/in:	Philip Steinlein & Bianca Knittel
Author: Bianca Knittel & Philip Steinlein	Team: Just Go IT