

Ein Ersatz für NRPE und NSCA?

Sven Velt
team(ix) GmbH, Nürnberg

sv@teamix.net

- Linux seit »10 Jahren
- 1995-2001 Internet-Provider
 - Verantwortlich für Technik
 - 24/7 muss alles laufen
 - Monitoring notwendig
 - ... mehr als einmal nachts aufgestanden
- Seit 2002 bei team(ix) GmbH
 - Schulungen, u. a. **Nagios**, Apache, Samba
 - Projekte

- Gegründet 2001
- Ursprünge: Open-Source und Netzwerke
- Heute auch:
 - NetApp
 - VMWare
 - Riverbed (WAN-Beschleunigung)
 - Juniper
 - N-IX (Nürnberger Internet-eXchange)

- <http://www.teamix.net/teamix/jobs/>
- Wir suchen:
 - OSS / Nagios – Consultant und Trainer
 - VMWare – Consultant und Trainer
 - NetApp – Consultant und Trainer
 - Azubi ;-)

- Partner von team(ix)
- 15 Schulungsräume hier im SWP
- Breites Themen-Spektrum:
 - Storage
 - Virtualisierung
 - Betriebssysteme
 - Applikationen
- Besonderheit: Brücken-Workshops
- Führung in der Mittagspause ;-)

- Motivation
- Ideen
- Umsetzung
- Die Bastel-Lust aka „Was gibt's schon?“
 - Was kommt noch?
- Wenn die Zeit reicht: Demo

- Zwei Kunden möchten je eine Linux-Maschine
 - Apache als Reverse-Proxy (Diversifikation)
 - (Nahezu) einzige Linux-Maschine im Unternehmen
 - Deswegen: (Security-)Support, Maintenance
 - Monitoring: u.a. Security-Updates, RAID, ...
- Problem: Kein dauerhafter Zugriff per SSH
 - Beide Rechner hinter Firewall
 - Wartungs-VPN wird manuell hochgefahren

- Ansatz 1: NRPE / SSH
 - Kein Zugriff durch Firewall gewünscht
- Ansatz 2: NSCA
 - Möglichkeit bei team(ix), den NSCA/Nagios via Internet zu „füttern“ - ***brr***
- Ansatz 3: Nagios auf beiden Maschinen

- Zugriff per HTTP auf Port 80 möglich
 - Apache läuft
 - Keine Änderung an Firewall
 - „versteckte“ URL enthält Informationen bzw. Check-Ergebnisse
 - Verschlüsselung (HTTPS) und Authentifizierung (HTTP Authentication) möglich

- Reverse-Proxy / Web-Server beim Kunden
 - Kleines Skript per Cron ausgeführt
 - Führt Plugin(s) aus
 - Schreibt Ergebnisse in Datei
- team(ix)-Nagios
 - Skript per cron oder Nagios ausgeführt
 - Holt Datei von Web-Server
 - Schreibt Ergebnisse in Nagios (z.B. als passive Events)

- Aufbau: (einfaches) XML

BASE

- timestamp?
- host+
 - name
 - returncode?
 - output?
 - timestamp?
- service+
 - description
 - returncode
 - output
 - timestamp?

- Hostname, Service Description und Plugin Output BASE64-encoded
 - Unterschiedliche Lokalisierung/Kodierung auf Quelle und Ziel **macht** Probleme
 - Leicht auch in Shell-Skripten zu erzeugen

```
% echo 'Hallo, Sven!' | recode data..base64  
SGFsbG8sIFN2ZW4hCg==
```

- Client-Maschine:
 - Ausführen der Plugins
 - Erzeugen der XML-Datei
- Nagios-Maschine:
 - Analyse der XML-Datei
 - Ergebnisse an Nagios weitergeben
- Fehlt: Der Transport

- **Ich liebe HTTP**
 - Einfaches Protokoll
 - Wird überall gesprochen („wget“, „curl“)
 - Kann über (Reverse-)Proxy weitergeleitet werden
 - Verschlüsselung und Authentifizierung eingebaut
 - Filterung möglich („Virus-Wall“)
 - Aufbrechen des HTTPS an Virus-Wall
 - Check
 - (interne) Weiterleitung per HTTP oder HTTPS

- Ursprünglich:
 - „Dirty shell scripts“ beim Kunden
 - Python-Skript auf Nagios-Server

- Heute: **Nag(ix)SC**
 - Alles in Python (2.4 - 2.6)
 - „Saubere“ XML-Erstellung bzw. -Analyse
 - Kann deutlich mehr, dazu später mehr ;-)

- Anlegen einer „conf“-Datei

```
[nagixsc]
Reserved: for future use
```

```
[host1]
_section_name: Hostname in Nagios
_host_check: .../check_host -H 127.0.0.1
Disk_Root: .../check_disk -w10% -c5% -p /
```

```
[host2]
_host_name: host2.nagios
Swap: .../check_swap -w50% -c25%
```

- Ausführen der Checks

```
% ./nagixsc_conf2xml.py -c NagPortWS.conf
```

```
<?xml version="1.0"?>  
<nagixsc version="1.0">  
<timestamp>1274871405</timestamp>  
<host>  
  <name>aG9zdDE=</name>  
  <service>  
    <description> [...] </description>
```

```
% ./nagixsc_conf2xml.py -c NagPortWS.conf  
                        -o NagPortWS.xml
```

- Achtung! Reines Debugging! Aufruf:

```
% ./nagixsc_read_xml.py -f NagPortWS.xml
```

```
Host:      host1
Service:   None
RetCode:   0
Output:    'OK - 127.0.0.1 responds to ICMP. [...]'
Timestamp: 1274871701
```

```
[...]
```

```
Host:      host2.nagios
Service:   Swap
RetCode:   0
Output:    'SWAP OK - 97% free (1838 MB out [...])'
Timestamp: 1274871702
```

- Einfach nachvollziehbar als passive Checks:

```
% ./nagixsc_xml2nagios.py -f NagPortWS.xml  
-O passive  
-p /dev/stdout
```

```
[1274872674] PROCESS_HOST_CHECK_RESULT;host1;0;OK [...]
```

```
[1274872674] PROCESS_SERVICE_CHECK_RESULT;host1;  
Disk_Root;0;DISK OK - [...]
```

```
[1274872674] PROCESS_SERVICE_CHECK_RESULT;host2.nagios;  
Swap;0;SWAP OK - [...]
```

```
% ./nagixsc_xml2nagios.py -f NagPortWS.xml  
-O passive
```

- Somit ursprüngliche Aufgabe gelöst
 - `nagixsc_xml2nagios.py` spricht auch HTTP(S) inkl. Authentifizierung
 - „-u URL“
 - „-l HTTP-User“
 - „-a HTTP-Passwort“
 - Ebenso Test auf veraltete Checks („timestamp“ im XML) möglich und setzen des Returncodes auf UNKNOWN
 - „-m“ (Mark old checks)
 - „-s SECONDS“

- Was machen NRPE und NSCA?
 - Check-Ergebnisse transportieren
 - NRPE
 - „Abholen“ der Ergebnisse von außen bzw. durch Nagios-Server
 - NSCA
 - Übermitteln der Ergebnisse zum Nagios

- Konfigurationsdatei („sample-configs/conf2http.cfg“):

```
[server]
```

```
ip: 0.0.0.0
```

```
port: 15666
```

```
ssl: false
```

```
sslcert: server.pem
```

```
conf_dir: ./sample-configs/conf
```

```
[users]
```

```
; echo -n "Password" | md5sum -
```

```
nagixsc: 019b0966d98fb71d1a4bc4ca0c81d5cc ; PW: nagixsc
```

- Aufruf:

```
% ./nagixsc_conf2http.py -c ../conf2http.cfg
```

- Weitere Parameter:

- „-d“: (daemon) Als Dienst im Hintergrund starten
- „- - noSSL“: Kein SSL, überschreibt cfg-Datei

- Daten abholen:

```
% ./nagixsc_xml2nagios.py  
    -u http://localhost:15666/nagixsc/  
    -l <user> -a <password> -0 <mode>
```

- „<mode>“ kann sein:

- „passive“, „passive_check“
- „checkresult“, „checkresult_check“
- „active“
- („multi“)

- Die URL setzt sich zusammen aus:
 - Protokoll, Hostname, Port – klar ;-)
 - Name der „.conf“-Datei (ohne Endung) innerhalb des als „conf_dir“ angegebenen Verzeichnisses
 - Optional: Hostname bzw. Section/Abschnitt
 - Optional: Service Description

```
http://localhost:15666/nagixsc/
```

```
http://localhost:15666/nagixsc/host1/
```

```
http://localhost:15666/nagixsc/host2/Swap/
```

- Konfigurationsdatei („sample-configs/http2nagios.cfg“):

```
[server]
```

```
ip: 0.0.0.0
```

```
port: 15667
```

```
ssl: false
```

```
sslcert: server.pem
```

```
mode: checkresult
```

```
[mode_checkresult]
```

```
dir: /.../checkresults
```

```
[users]
```

```
nagixsc: 019b0966d98fb71d1a4bc4ca0c81d5cc ; PW: nagixsc
```

- Aufruf:

```
% ./nagixsc_http2nagios.py -c ../http2nagios.cfg
```

- Weitere Parameter:

- „-d“: (daemon) Als Dienst im Hintergrund starten
- „- - noSSL“: Kein SSL, überschreibt cfg-Datei

- Daten senden:

- Mit Hilfe von „nagixsc_conf2xml“:

```
./nagixsc_conf2xml.py -c NagPortWS.conf  
-o http://127.0.0.1:15667/  
-l <user> -a <password>
```

- Mit Hilfe von z.B. „wget“ oder „curl“:

```
curl -u <user>:<password>  
-F 'xmlfile=@<xml-filename>'  
http://127.0.0.1:15667/
```

- Filtern bei conf2xml und xml2nagios
 - Nach Host (XML) bzw. Abschnitt (conf-Datei)
 - Nach Service (beides)
- XML-Syntax testen
- Läuft prinzipiell auch unter Windows
- Kleine Hacks:
 - `nagixsc_xml2cfg`
 - `(nagixsc_live2xml)`

- HTTP-Server
 - ACLs
 - SSL Checks und SSL Client Zertifikate
- `nagixsc_conf2http`
 - Remote-Admin der „*.conf“-Dateien
 - Eigener Scheduler / Caching der Ergebnisse
- Python 2.6 und bundled SSL (Windows!)

- Optimierung der „*.conf“-Dateien
 - Services auf mehreren Hosts
 - Variablen („\$HOSTADDRESS\$“)
- Python 2.3 (z.B. SLES 9 – und alle so „yeah“...)
- Dokumentation *hust*

- Quellcode zur Zeit unter
 - <http://github.com/wAmpIre/nagixsc/>
- Ansonsten:
 - Demo

Sven Velt, team(ix) GmbH, sv@teamix.net