softScheck
we identify vulnerabilities others don't

Robot Operating System (ROS): Safe & Insecure

Sicherheitsuntersuchung des Robot OS (ROS)

Stand: 23. Juni 2014

Lubomir Stroetmann
Prof. Dr. Hartmut Pohl
softScheck GmbH Köln

Inhaltsverzeichnis

Security (Angriffssicherheit) von Robotersystemen ist unabdingbar, da sie die Safety (Betriebssicherheit) beeinträchtigt. So wird zwar bei Robotern generell auf die Safety geachtet und es werden Schutzmechanismen implementiert, um z.B. zu verhindern, dass ein Roboter einen Menschen verletzt, allerdings kann ein Angreifer durch Ausnutzen einer Sicherheitslücke in Software implementierte Schutzmechanismen leicht außer Kraft setzen und den oder die angegriffenen Roboter fehlsteuern.

Die Bundesregierung wird die Haftung der IT-Hersteller und -Diensteanbieter für Datenschutz- und IT-Sicherheitsmängel in den nächsten beiden Jahren gesetzlich regeln. Dies betrifft insbesondere Industriesteuerungen und Sicherheitssoftware wie industrielle Firewalls, Verschlüsselung etc.

ROS wurde daher einer Sicherheitsprüfung unterzogen und unter Berücksichtigung der Sicherheitsziele Vertraulichkeit, Integrität, Verfügbarkeit und Authentizität bewertet. Dazu wurde die Funktionsweise aller ROS-Komponenten betrachtet und auf die Erfüllung der Sicherheitsziele hin untersucht. Zusammenfassend ist festzustellen, dass ROS als relativ junges System weder beim Design noch bei der Implementierung Aspekte der Security berücksichtigt, auf der Basis von ROS entwickelte Systeme sind daher völlig unsicher.

Das Robot Operating System (ROS) ist ein Open-Source Softwareframework zur Entwicklung, Steuerung und Kommunikation von und mt Robotern. ROS enthält Tools wie eine 3D-Simulationsumgebung und Bibliotheken, die die Erstellung von komplexen Robotersystemen mit Funktionalitäten wie z.B. Navigation, Gelenkbewegungen etc. unterstützen. ROS ist modular aufgebaut, so dass sowohl einzelne Roboterkomponenten als auch ganze Roboter miteinander über TCP/IP kommunizieren können. Es wird von der Open Source Robotics Foundation (OSRF) unter der BSD-Lizenz als Open Source zur Verfügung gestellt und ist gut dokumentiert, unter anderem verfügt es über ein eigenes Wiki (http://wiki.ros.org/).

1 Funktionsweise von ROS

ROS ist ein auf IP-basierendes modulares Kommunikationsframework für Roboter und deren Komponenten wie Sensoren und steuernde Motoren. Über hardwarespezifische Module wird eine Abstraktion von der Roboterhardware erreicht, so dass auch Roboter und Roboterkomponenten verschiedener Hersteller verbunden werden können und Hersteller- und Hardware-unabhängig programmiert wird. Dazu wird jede Roboterkomponente als Node implementiert und läuft als eigener Prozess. Da die Kommunikation auf IP basiert, sind Komponenten und Roboter beliebig trennbar – so können z.B. Roboter an einer Fließbandstraße miteinander über ein WLAN kommunizieren und es können sogar Roboter weltweit über das Internet Kontakt aufnehmen.

Die Kommunikation zwischen Komponenten kann sowohl synchron als auch asynchron erfolgen. Synchrone Kommunikation, bei der Sender und Empfänger aufeinander warten, erfolgt über ‚Services', asynchrone Kommunikation erfolgt über ‚Topics'.

Die Programmierung von ROS kann in mehrere Programmiersprachen erfolgen. Zurzeit sind Programmbibliotheken für Python, C++, Lisp, Java und Lua implementiert und benutzbar.

Im Folgenden werden die wichtigsten Konzepte von ROS erläutert.

ROSCore

Die zentrale Server-Komponente von ROS wird als ROSCore bezeichnet. Dieser Core besteht aus drei Prozessen:

- Der ROS Master verwaltet alle Nodes, Topics und Services. Nodes melden sich am Master an und können Informationen über andere Nodes anfragen. Somit dient der Master als zentrales Register welches Nodes ermöglicht, sich gegenseitig zu finden. Sobald dies geschehen ist, kommunizieren Nodes direkt miteinander.
- Der ROS Parameter Server dient zum Speichern von Konfigurationsparametern der am Master angemeldeten Nodes. Parameter lassen sich zur Laufzeit ändern.
- Rosout ist ein Node, der von ROSCore zu Logging-Zwecken zur Verfügung gestellt wird.

Nodes

Jeder ROS-Prozess wird als Node bezeichnet. So ist z.B. jeder Sensor und jeder Motor eines Roboters als eine Node erreichbar. Nodes kommunizieren untereinander über Topics und Services. Verbindungen zwischen Nodes können mit Hilfe des ROS-Graphen visualisiert werden.

Messages

Messages sind Datenstrukturen, die der Kommunikation in ROS dienen. Messages werden von Nodes in Topics gepostet und abonniert. Messages können auch geloggt oder in Dateien (‚Bags') gespeichert werden. Eine Message ist eine Ansammlung verschiedener Datenfelder, die sich je nach Einsatzzweck unterscheiden.

Topics

Topics erlauben asynchrone Kommunikation zwischen ROS-Nodes. Nodes können Messages in Topics posten (‚publication') und Topics abonnieren (‚subscription'), um dort gepostete Messages zu empfangen. Eine Node kann gleichzeitig mehrere Topics abonnieren und kann in beliebig viele Topics posten.

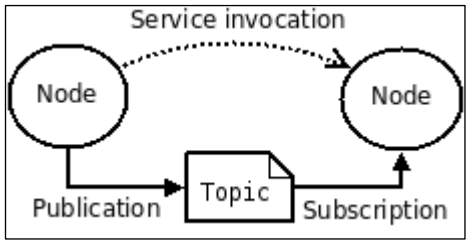

Abb. 1: Node-Kommunikation über ein Topic

Services

Services werden von Nodes angeboten und ermöglichen eine synchrone Anfrage-Antwort-Interaktion zwischen Nodes. Ein Node startet einen Service unter einem bestimmten Namen und dient damit als Server, dem ein anderer Node Anfragen schicken kann.

ROS Computation Graph

Der Computation-Graph visualisiert die Kommunikation unter den Komponenten des ROS-Systems zur Laufzeit.

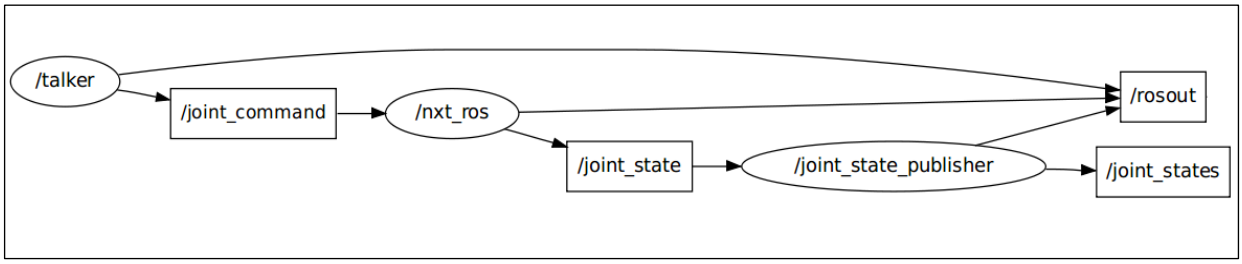

Abb. 2: ROS Computation Graph eines einfachen Roboters

Der Computation-Graph kann durch den Befehl rxgraph angezeigt werden und zeigt Nodes als Ellipsen und deren Topics als Rechtecke.

2 Sicherheitsanalyse

Die Funktionsweise aller ROS-Komponenten wurde analysiert und auf die Erfüllung der grundlegenden Sicherheitsziele Verfügbarkeit, Integrität, Vertraulichkeit und Authentizität hin untersucht. Zusammenfassend ist festzustellen, dass ROS als relativ junges (die Entwicklung von ROS begann 2007) und hauptsächlich in Forschungslaboren von Universitäten eingesetztes System weder beim Design noch bei der Implementierung Aspekte der Security bedacht hat.

2.1 Verfügbarkeit

Verfügbarkeit als Sicherheitsziel fokussiert sich auf die Verhinderung oder zumindest Verminderung von Systemausfällen als Folge von Angriffen. Systeme müssen robust genug gebaut sein, um Angriffen standzuhalten, die auf die Überlastung des Systems mit Daten oder die Überlastung durch Ausnutzen aller Systemressourcen abzielen. Systeme sollten Endlosschleifen erkennen und abbrechen, die Verarbeitung übermäßig großer Datenmengen nach einem festgelegten Zeitpunkt stoppen, bei Ausfällen sich selbständig neustarten und möglichst redundant ausgelegt sein, um die Dauer eines Ausfalls überbrücken zu können.

ROS bietet bezüglich der Verfügbarkeit ein akzeptables Grundniveau, indem sich der ROSCore bei Absturz oder Ausfall selbständig neu startet. Durch das ROS-Modul Multimaster können mehrere ROS Masters betrieben werden und damit redundante Systeme ohne den Core als Single Point of Failure aufgebaut werden. Allerdings ist ROS gegen komplexere logische Angriffe, wie in Abschnitt 4 beschrieben, nicht geschützt.

Über die Ausfallsicherheit der Nodes lässt sich keine Aussage treffen, da diese bei dem Bau eines

spezifischen Roboters von den Entwicklern getrennt programmiert werden. Deren Ausfallsicherheit hängt damit direkt von der Implementierung ab.

2.2 Integrität

Integrität bedeutet die Sicherstellung der Korrektheit (Unversehrtheit, Richtigkeit und Vollständigkeit) von Daten. Alle Daten, die verarbeitet, übertragen und gespeichert werden, dürfen nur mit ausdrücklicher Berechtigung modifiziert werden können. Dies geht über die Anwendung einfacher Prüfsummen hinaus, da eine Prüfsumme selbst auch modifiziert werden kann.

Da ROS auf einem Grund-PC-Betriebssystem aufgesetzt wird (derzeit bevorzugt Ubuntu Linux - rudimentäre Unterstützung von Windows und Mac OS X ist verfügbar), ist die Verarbeitung und Speicherung von Daten auf das Betriebssystem ausgelagert, welches Mechanismen zur Sicherstellung der Datenintegrität zur Verfügung stellt. Im Folgenden wird daher nur die Übertragung von Daten durch ROS über das Netzwerk betrachtet.

Die Datenübertragung bei ROS ist IP-basiert und gliedert sich in zwei Kommunikationskanäle:

- Kommunikation zwischen ROS Master und Nodes. Diese erfolgt nach dem XML/RPC-Standard, welcher XML-Daten über HTTP verschickt.
- Kommunikation von Nodes untereinander. Diese erfolgt wahlweise über TCP oder UDP und verwendet Binärdaten, die nach dem TCPROS bzw. UDPROS Protokoll kodiert sind.

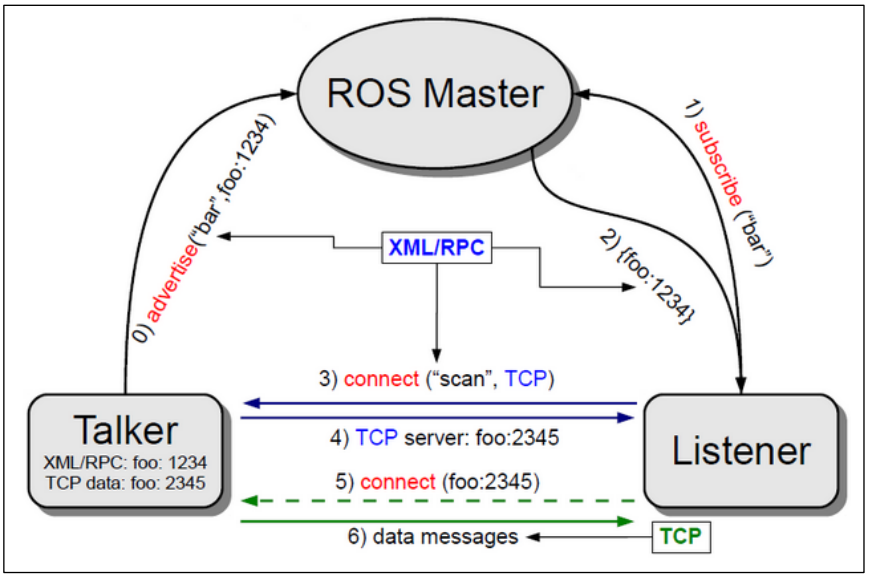

Abb. 3: Datenübertragung bei ROS

Durch die in TCP vorhandenen Prüfsummen ist eine triviale Datenintegrität gegeben, die aber für die Betrachtung der Sicherheit nicht ausreichend ist. TCPROS enthält eine MD5-basierte Prüfsumme für den Nachrichtentyp, allerdings nicht für die Daten.

Somit ist die Integrität von Daten in ROS nicht gegeben. Sämtliche Daten werden unverschlüsselt übertragen und können von jedem Angreifer im Netzwerk manipuliert werden um damit z.B. einen Roboter zu ungewollten Bewegungen zwingen oder Sensordaten verfälschen.

2.3 Vertraulichkeit

Vertraulichkeit umfasst den Schutz gegen die unberechtigte Kenntnisnahme von Daten bei deren Verarbeitung, Speicherung und Übertragung.

Bei der Übertragung von Daten über das Netzwerk ist bei ROS keinerlei Vertraulichkeit gegeben. Die Daten werden unverschlüsselt übertragen und können mitgeschnitten und gelesen werden. So kann ein Angreifer alle Sensordaten und alle Befehle abhören und auf Grund der fehlenden Authentifizierung auch wieder einspielen.

Die Verwendung zusätzlicher Sicherheitsmechanismen wie z.B. ein physisch abgeschottetes Netzwerk, ein verschlüsseltes WLAN oder eine Internetverbindung über gesicherte VPNs das grundlegende Problem nicht löst. WLANs und VPNs können gehackt werden, physisch gesicherte Einheiten aufgebrochen werden. Die komplett modulare und IP-basierte Struktur von ROS fördert zudem die starke Vernetzung über das Internet. Fernsteuerung oder die autonome Kommunikation von Robotern an verschiedenen Orten ist möglich.

Die Speicherung von Daten unterliegt dem ROS zugrundeliegenden Betriebssystem. Dadurch kann die Vertraulichkeit der gespeicherten Daten entsprechend durch Festplattenverschlüsselung geregelt werden.

2.4 Authentizität

Bei Security-relevanten Aktivitäten müssen die Kommunnikationsteilnehmer identifiziert und authentifiziert werden. ROS verfügt über keinerlei Authentifizierungsmechanismen. Jeder kann und darf mit jedem kommunizieren, jedem Teilnehmer wird implizit vollständig vertraut. Ein Angreifer kann sich als beliebiger Node ausgeben und diesen imitieren. Es gibt kein Zugriffskontrollsystem, das festlegt und kontrolliert, wer welche Art von Aktionen durchführen darf. So ist es z.B. jedem erlaubt, einen Shutdown-Befehl an einen Node oder sogar an den ROSCore zu schicken. Dieser Befehl wird gar nicht hinterfragt sondern sofort ausgeführt, was zum Stillstand des Systems führt und somit die Verfügbarkeit verletzt. Das Python-Skript in Abb. 4 veranschaulicht diesen Angriff, indem es nicht-authentifiziert dem ROSCore, der auf TCP Port 11311 läuft, mittels XML-RPC einen Shutdown-Befehl sendet.

```
#!/usr/bin/env python
import os
import xmlrpclib
s = xmlrpclib.ServerProxy('http://roscore:11311', allow_none=True,verbose=False)
caller_id="test"
print s.shutdown(caller_id, "und weg")
```

Abb. 4: Shutdown-Skript in Python

Der ROSCore schaltet sich darauf hin ab, wodurch keine neue Kommunikation zwischen Nodes mehr aufgebaut werden kann. Der Inhalt des vom Skript generierten HTTP-Pakets mit XML-Inhalt (s. Abb. 5) kann auf Grund der fehlenden Vertraulichkeit abgehört und manipuliert werden.

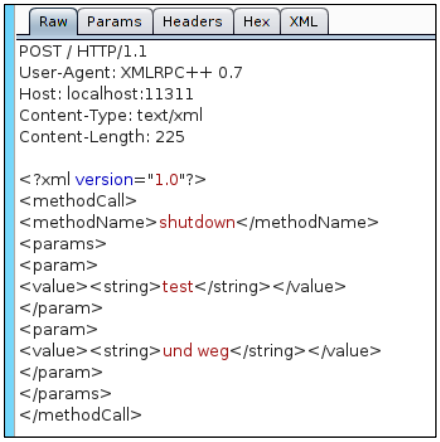

Abb. 5: XMLRPC Shutdown-Befehl

Durch das Fehlen von Authentifizierung, Identitäten und Berechtigungen besteht auch keine Möglichkeit, eine Hierarchie von Roboterkomponenten zu implementieren, in der z.B. eine Steuereinheit einen Motor steuert, aber nicht umgekehrt der Motor Befehle an die Steuereinheit erteilen darf.

In einem Labor- und Forschungsumfeld ist die freie, unkontrollierte Kommunikation vielleicht vertretbar. Für die Sicherheitsvoraussetzungen in einem Industrie- oder Produktionsumfeld reichen die von ROS zur Verfügung gestellten Kommunikationsmittel allerdings nicht aus. Die Kommunikation zwischen Nodes und Master könnte, da sie auf HTTP basiert, recht einfach durch die Verlegung auf mit TLS gesichertem HTTPS gesichert werden. Die Kommunikation zwischen Nodes über TCPROS bzw. UDPROS müsste allerdings grundlegend überarbeitet werden.

3 Sicherheitslücken

Bei der Untersuchung von ROS und der Kommunikation zwischen ROS-Komponenten sind neben den im vorigen Abschnitt diskutierten fehlenden Sicherheitszielen auch Sicherheitslücken aufgefallen.

3.1 Doppelte und anonyme Nodes

ROS unterliegt einem Design-Fehler bei der Initialisierung von Nodes. Wird ein Node mit einem Namen, der schon belegt ist, gestartet, so zwingt der ROS Master die alte Node zum Herunterfahren. Dies soll vermeiden, dass zwei Nodes mit gleichem Namen existieren. Ein Angreifer kann so andere Nodes zum Herunterfahren zwingen, indem er sich zunächst vom ROSCore eine Liste aller aktiven Nodes ausgeben lässt und dann sukzessiv neue funktionslose Nodes mit den gleichen Namen startet. Laufende Nodes werden somit beendet und die Funktion des Roboters komplett stillgelegt.

ROS erlaubt zusätzlich das Erstellen von „Anonymous"-Nodes, die auch identische Namen tragen können und auf die diese Sicherheitslücke nicht zutrifft. Die Standardeinstellung für die Erstellung von Nodes ist allerdings die Identifizierung über Namen.

3.2 XML Denial of Service

Zur Kommunikation zwischen ROS Master und Nodes wird XML-RPC benutzt, ein auf XML und HTTP basiertes Protokoll. Entsprechend unterliegt ROS ein XML-Parser, der die empfangenen XML-Befehle interpretiert (Python-Paket xmlrpc). Der XML-Standard umfasst weitreichende Möglichkeiten, wie z.B. die Einbindung externer Dokumente. XML verwendende Systeme sind XML-spezifischen Angriffen ausgesetzt.

Ein häufiges Problem in XML-Parsern bilden XML-Entitäten, eine Art Textbaustein oder Konstante in XML. Eine Entität wird einmal mit einem Namen definiert und kann dann beliebig oft mit `&name;` aufgerufen werden, wobei der Name durch den eigentlichen Wert der Entität ersetzt (expandiert) wird. Dies ermöglicht die Konstruktion von sogenannten XML-Bomben, bei denen unter Ausnutzung der Expansion von Entitäten mit einer kleinen Eingabe eine enorm rechen- und speicherintensive Ausgabe erzeugt wird. Dies hat meist zur Folge, dass der Arbeitsspeicher des Systems voll läuft und das Programm abstürzt. Variationen der XML-Bombe sind die Definition einer Entität mit großem Inhalt, die sehr oft aufgerufen wird („Quadratic Blowup", der Rechenaufwand O(n) wächst quadratisch) oder die rekursive Definition vieler Entitäten, die die jeweils vorhergehende Entität mehrmals aufrufen (als „Billion Laughs" bekannt, der Rechenaufwand wächst exponentiell).

Die in Abb. 6 gezeigte Billion Laughs XML-Bombe erzeugt mit einem XML-Code, der weniger als 1 kB groß ist, eine Ausgabe von ca. 3GB, deren Berechnung lange dauert und viel Arbeitsspeicher belegt. Der Angriff kann trivial vergrößert werden. Schickt man diese XML-Bombe an den ROS Master oder an eine Node, so führt dies zum Volllaufen des Arbeitsspeichers und zum Absturz des Prozesses. Ein Angriff auf Nodes ist effektiver, da sich der ROS Master selbständig neu startet.

```
<?xml version="1.0"?>
<!DOCTYPE bomb [
  <!ELEMENT bomb (#PCDATA)>
  <!ENTITY a "aaaaaaaaaa">
  <!ENTITY b "&a;&a;&a;&a;&a;&a;&a;&a;&a;&a;">
  <!ENTITY c "&b;&b;&b;&b;&b;&b;&b;&b;&b;&b;">
  <!ENTITY d "&c;&c;&c;&c;&c;&c;&c;&c;&c;&c;">
  <!ENTITY e "&d;&d;&d;&d;&d;&d;&d;&d;&d;&d;">
  <!ENTITY f "&e;&e;&e;&e;&e;&e;&e;&e;&e;&e;">
  <!ENTITY g "&f;&f;&f;&f;&f;&f;&f;&f;&f;&f;">
  <!ENTITY h "&g;&g;&g;&g;&g;&g;&g;&g;&g;&g;">
  <!ENTITY i "&h;&h;&h;&h;&h;&h;&h;&h;&h;&h;">
  <!ENTITY j "&i;&i;&i;&i;&i;&i;&i;&i;&i;&i;">
  <!ENTITY k "&j;&j;&j;&j;&j;&j;&j;&j;&j;&j;"> ]>
<bomb>&k;</bomb>
```

Abb. 6: "Billion Laughs" XML-Bombe

Um sich gegen XML-Bomben zu schützen empfiehlt sich die Verwendung des defusedxml-Pakets:

https://pypi.python.org/pypi/defusedxml#python-xml-libraries

Weiterführende Links zum Thema:

http://de.wikipedia.org/wiki/Entitätsexpansion

http://docs.python.org/2/library/xml.html#xml-vulnerabilities

4 Fazit

ROS bietet eine sehr mächtige Kommunikationsplattform für Roboter, bei der allerdings keinerlei Sicherheitsaspekte bedacht worden sind. Dies ist zum einen fatal, da die Betriebssicherheit eines Robotersystems von der Angriffssicherheit abhängt. Ein Angreifer, der durch seinen Angriff Schutzmechanismen des Roboters z.B. gegen falsche Bewegung, Überhitzung o.ä. ausschaltet, kann enormen Schaden verursachen. Zum anderen ermöglicht ROS die ungesicherte Kommunikation von Robotern und Roboterkomponenten über Netzwerke und das Internet. Dadurch entsteht eine Angriffsfläche, in der jeder Betreiber von ROS-kontrollierten Robotern, der keine zusätzlichen Schutzmechanismen benutzt, angreifbar ist. Die Effektivität eines jeden zusätzlich aufgesetzten Schutzmechanismus hängt ab von der Expertise des Anwenders.

Vor einer im Unterprojekt „ROS Industrial" vorgesehene Transition aus dem Forschungs- und Laborbetrieb in die Industriepraxis empfiehlt sich ein gründliches Re-design der ROS Konzepte unter Berücksichtigung der Sicherheitsziele Vertraulichkeit, Integrität, Verfügbarkeit und Authentizität. Für die Implementierung von Sicherheitsmechanismen wie Authentifizierung und Verschlüsselung stehen fertig verfügbare Standardmechanismen (z.B. HTTPS statt HTTP) und Bibliotheken (z.B. OpenSSL) bereit, auf die zurückgegriffen werden kann.

5 Literaturverzeichnis

Open Source Robotics Foundation (OSRF) (Hrsg.):	ROS Wiki. Mountain View 2013 http://wiki.ros.org/
McClean et al.:	A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS). Cambridge 2013
Python Software Foundation (Hrsg.):	XML Vulnerabilities. Wilmington 2014 http://docs.python.org/2/library/xml.html#xml-vulnerabilities
Heimes, C.:	defusedxml 0.4.1. o.O. 2014 https://pypi.python.org/pypi/defusedxml#python-xml-libraries

softScheck
we identify vulnerabilities others don't

Prof. Dr. Hartmut Pohl
Geschäftsführender Gesellschafter
softScheck GmbH Köln www.softScheck.com

Büro: Bonner Str. 108. 53757 Sankt Augustin

Tel.: +49 (2241) 255 43 - 12
Mobil: +49 (172) 9437 - 329
Fax: +49 (2241) 255 43 – 29
Hartmut.Pohl@softScheck.com