Technische Universität Dresden

Fakultät Informatik

Professur für Datenschutz und Datensicherheit

Projekt HoneySens

10. Zwischenbericht

Forschungsprojekt zur Entwicklung eines Sensornetzwerks, um Angriffe auf die IT-Infrastruktur des Freistaates zu erkennen

Stand: 30.06.2018

Autor: Dipl.-Inf. Pascal Brückner
Betreuung: Dr.-Ing. Stefan Köpsell

1 Einleitung

Der Beauftragte für Informationssicherheit des Landes Sachsen prüft zur Verbesserung der IT-Sicherheit des Sächsischen Verwaltungsnetzes (SVN) die Grundlagen zum Aufbau eines Sensornetzwerkes, mit dem verdächtige Zugriffe auf Netzwerkdienste und -geräte erkannt und zuständige Stellen zeitnah informiert werden können. Das geplante System soll sich insbesondere durch eine leichte Bedienbarkeit und gute Skalierbarkeit auszeichnen. Hauptaugenmerk liegt dabei auf den Funktionalitäten, die die Wartung und Verwaltung der Sensorinfrastruktur betreffen. Weitere Faktoren, darunter die möglichst einfache und transparente Integration der zu entwerfenden Architektur in das bestehende Netzwerk, sowie ein autonomer, von anderen Netzwerkkomponenten und -diensten unabhängiger Betrieb sind ebenfalls zu berücksichtigen.

In den vergangenen Monaten ist es gelungen, in Zusammenarbeit mit der T-Systems MMS die in den vorherigen Berichten angekündigte HoneySens Version 1.0 und die damit verbundenen Meilensteine fertigzustellen. Dieser Bericht gibt einen Überblick über die wichtigsten Neuerungen und Entwicklungen, die im Zuge dieses Prozesses entstanden sind. Die Anforderungen für diese finale Revision 1.0 umfassten neben diversen kleineren Anpassungen an der Benutzeroberfläche und dem Sensorverhalten insbesondere den Wunsch nach virtualisierten Sensoren und der Modularisierung der Honeypot-Dienste. Viele Vorarbeiten in diese Richtung wurden im vergangenen Jahr erledigt, so dass beispielsweise die Grundlagen für modulare Dienste sowohl konzeptionell als auch in Codeform bei Beginn dieser Projektphase bereits gelegt waren.

Eine unerwartete Herausforderung war die im Laufe der Projektphase zusätzlich hinzugekommene Anforderung, den HoneySens-Server auch mit Orchestrierungssystemen wie Kubernetes oder - im Falle des SVN - Docker Swarm einsetzen zu können. Mit der Umstellung auf die neue Infrastruktur des SVN2 war seitens der IT der Landesverwaltung vorgesehen, bestehende Docker-Instanzen auf ein neues Swarm-System zu migrieren. Der damit einhergehende Anforderungskatalog an externe Dienstleister beinhaltete eine ganze Reihe von Bedingungen, die der bestehende Server-Container nicht leisten konnte. Es war deshalb nicht zu vermeiden, im Januar einen Sprint einzulegen, um vor der Abschaltung der bestehenden Docker-Plattform des SVN1 eine saubere Migration auf das Swarm-System vornehmen zu können. Diese Phase war von diversen Schwierigkeiten und Problemen begleitet, auf die im entsprechenden Abschnitt genauer eingegangen werden wird.

Trotz des zeitlichen Drucks ist es letztendlich jedoch gelungen, das HoneySens-Netz im SVN2 mit der beabsichtigten Version 1.0 erfolgreich in Betrieb zu nehmen. Aufgrund der umfangreichen architekturellen Änderungen in diesem Meilenstein wurde zudem parallel die finale Revision 0.2.5 des Servers veröffentlicht, die als aktuelle Legacy-OpenSource-Variante des Systems auch über GitHub frei genutzt werden kann [1]. Die für diese Version angefallenen Arbeiten sind in diesem Dokument ebenfalls festgehalten.

Die im ersten Zwischenbericht beschriebenen Anforderungen behalten ihre Gültigkeit und gelten weiterhin als Richtlinie für zukünftige Entwicklungen. Sie werden an dieser Stelle nicht erneut ausgeführt.

[1] `https://github.com/sylencecc/HoneySens-Legacy`

2 Sachstand

Dieser Abschnitt gibt einen Überblick über die vom 01. Dezember 2017 bis zum 30. Juni 2018 verzeichneten Fortschritte im HoneySens-Projekt. Es werden zunächst allgemeine Neuerungen in der Projektstruktur und dem Repository vorgestellt, die alle HoneySens-Komponenten gleichermaßen betreffen. Anschließend wird detailliert auf die Änderungen an der Server-Software eingegangen. Neuerungen bezüglich der Sensoren, sei es im Hinblick auf die Management-Software, die plattformspezifischen Komponenten oder auch die nun dynamisch nachladbaren Honeypot-Dienste, werden im Anschluss beschrieben. Teilbereiche, die in den Anforderungen des ersten Projektberichtes genannt, aber hier nicht weiter ausgeführt werden, sind Gegenstand der zukünftigen Weiterentwicklung.

2.1 DevOps: Plattformunabhängige, reproduzierbare Entwicklungsumgebungen

HoneySens ist inzwischen zu einem Projekt und damit auch einer Codebasis von nicht unerheblicher Größe herangewachsen. Da seitens der MMS nun auch zusätzliche Entwickler für das Projekt bereitgestellt wurden, wurde der Ruf nach einer einheitlichen Entwicklungsumgebung und einem möglichst plattform- und systemabhängigen Buildprozess laut. Während zuvor alle Entwicklungssysteme mühsam händisch installierte und eingerichtete virtuelle Maschinen oder Container waren, galt es nun, diesen Prozess weitestgehend zu automatisieren und so die Arbeit mit dem Repository zu vereinfachen.

Ein erster Schritt in Richtung dieses Ziels war die Konsolidierung des Build-Prozesses: Alle Komponenten des Systems (Server, Sensorsoftware für verschiedene Plattformen und die Honeypot-Dienste) wurden mit Makefiles versehen, um auch auf einem nicht speziell präparierten Entwicklungssystem leicht die Software bauen zu können. Zusätzlich wurde darauf geachtet, externe Abhängigkeiten wie PHP- und JavaScript-Frameworks, Docker-Basisimages oder nicht in den Repositories vorhandene und somit direkt über GitHub o.ä. bezogene Komponenten auf bestimmte, garantiert kompatible Versionen zu fixieren. Somit ist auch in Zukunft, wenn von diesen Projekten neue Revisionen erscheinen, Kompatibilität mit der HoneySens-Codebasis sichergestellt. Die Idee hinter solchen *Reproducible Builds* ist, dass verschiedene Kompilate aus demselben Repository identische Ergebnisse erzeugen, unabhängig von den sonstigen Rahmenbedingungen des Systems, auf dem das Projekt kompiliert wird.

Durch den Zuwachs an Entwicklern, die ebenfalls zum Repository beisteuern, wurde es zudem unabdingbar, eine für alle Beteiligten einheitliche Entwicklungsumgebung zu schaffen. Der erste Schritt in Richtung dieses Ziels war das Erstellen eines speziellen Server-Docker-Images, das Entwickler bei ihrer Arbeit mit der API und der Webschnittstelle unterstützt. Es kann direkt mit Hilfe des Makefiles aus dem Repository erzeugt und gestartet werden, beinhaltet eine vollständige Server-Umgebung inklusive aller Voraussetzungen, um diese aus den Quellen neu zu erzeugen (darunter auch ein Buildsystem). Entwicklern ist es somit möglich, Änderungen an den Quelldateien vornehmen und deren Auswirkungen unmittelbar im Server-Container beobachten zu können. Um auch unter Microsoft Windows mit dieser Software arbeiten zu können, kommt *Vagrant*[2] zum Einsatz. Diese Software ermöglicht es, virtuelle Maschinen zu provisionieren und somit einheitliche Entwicklungsumgebungen unabhängig vom zugrundeliegenden Host-Betriebssystsem zu erschaffen. Die beschriebenen Maßnahmen gestatten es, aus dem Repository heraus einen Entwicklungsserver mit dem simplen Kommando `make dev` zu erzeugen. Die-

[2] `https://vagrantup.com`

ses generiert und startet im Hintergrund die nötigen Container und VMs, woraufhin der Server sofort einsatzbereit ist (vgl. Abb. 1). Ein Nutzer kann nun den lokalen Quellcode im Repository modifizieren und die Änderungen direkt im Server beobachten.

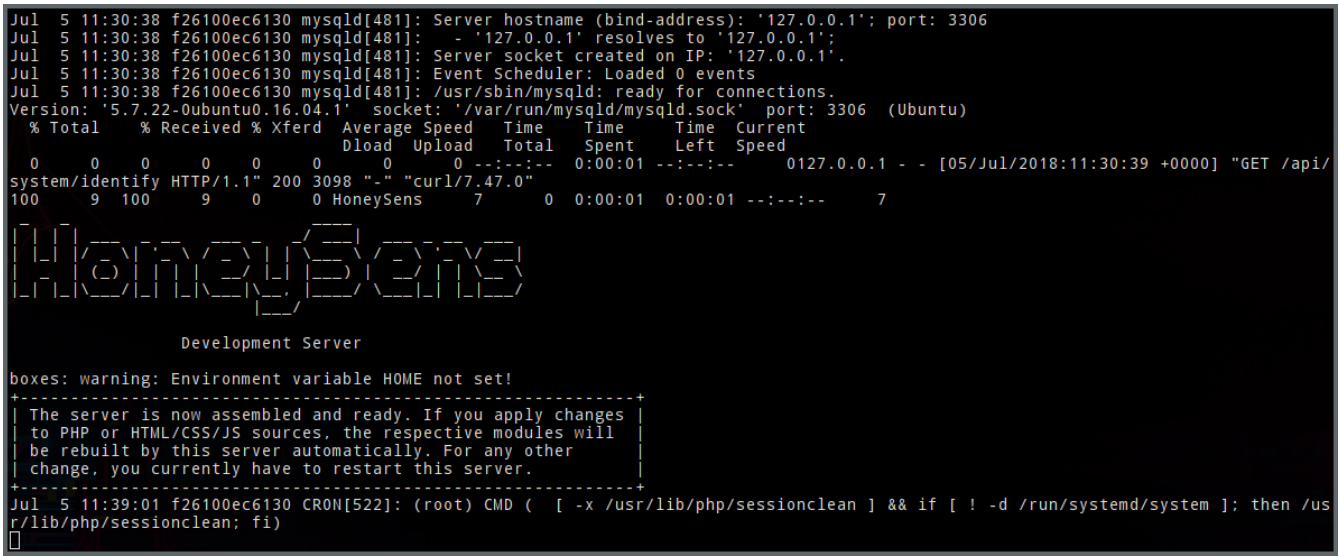

Abbildung 1: Startprozess des Entwicklungsservers

2.2 Server

Serverseitig lassen sich die Weiterentwicklungen primär in die Schwerpunkte Multi-Plattform-Support, Honeypot-Service-Registry, SVN2/Swarm-Support und operative Neuerungen einordnen. Diese sollen hier der Reihe nach näher beleuchtet werden.

Unterstützung verschiedener Sensor-Plattformen Die Möglichkeit, Sensoren nicht nur auf Basis der bewährten BeagleBone-Plattform betreiben zu können, wurde in den vergangenen Berichten bereits konzeptionell beleuchtet und bereits weitestgehend implementiert. In dieser Projektphase galt es lediglich, diese Arbeiten zum Abschluss zu führen und das System in dieser Konstellation umfassend zu testen. Zunächst wurde dazu das Konzept der architekturspezifischen „Plattformen“ als REST-Ressource in die API eingefügt. Die betreffenden Endpunkte werden von Sensoren genutzt, um die korrekte Firmware auf dem Server identifizieren und direkt herunterladen zu können. Im Frontend wurde hierfür ebenfalls ein neues Modul eingefügt, über das die bekannten Plattformen (derzeit BeagleBone Black und Sensoren auf Docker-Basis) eingesehen und Firmware für diese verwaltet werden kann (vgl. Abb. 2). Diese Oberfläche ersetzt vollständig das alte „Firmware“-Modul aus den Versionen 0.2.x. Analog zu diesen Veränderungen ist der globale Sensor-Updateintervall nun Bestandteil der allgemeinen Serverkonfiguration.

Weiterhin wurde die Administrationsoberfläche durch die Vereinheitlichung der Konfiguration von Sensoren weiter vereinfacht: Die bisherige Trennung von Sensor-Basiskonfiguration (Netzwerkeinstellungen, Name, Ort, Erreichbarkeit Server, Proxies usw.) und Sensor-Betriebseinstellungen (Update-Intervall, Ziel-Firmware, Honeypot-Dienste) sind nun Teil der Sensor-Übersicht und dem damit verbundenen Konfigurationsdialog. Das Modul „Sensor-Konfiguration“ entfällt somit und alle Modifikationen am Sensor-

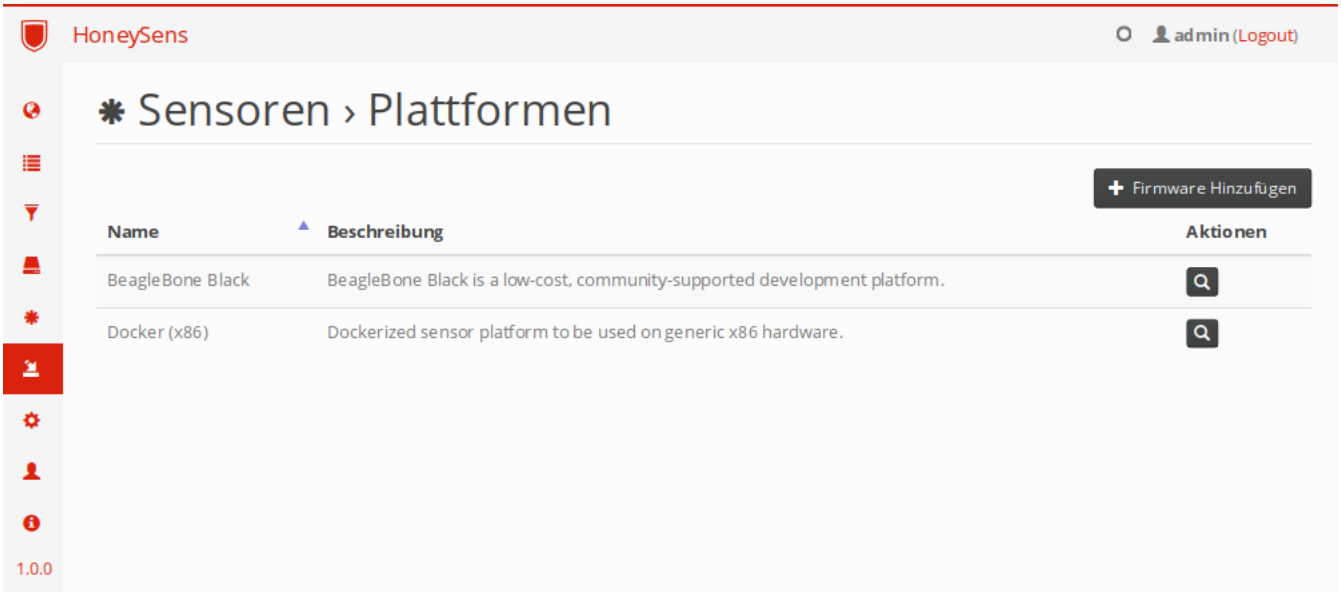

Abbildung 2: Plattform-Übersicht im Frontend

verhalten können jetzt an zentraler Stelle vorgenommen werden. Die Verteilung der modularen Honeypot-Dienste, die auf den Sensoren in Form von Docker-Containern betrieben werden, kann ebenfalls direkt mit Hilfe der Sensorübersicht angepasst werden. Diese beinhaltet nun eine Matrix mit allen verfügbaren Diensten resp. aller verfügbaren Sensoren, in der für jedes Tupel `(Sensor, Dienst)` bestimmt werden kann, ob der betreffende Honeypot-Dienst auf dem Sensor betrieben werden soll.

Im Zusammenhang mit der Umstellung auf die Swarm-Architektur für das SVN 2.0 waren seitens der Sensorsoftware massive Veränderungen im Umgang mit Proxy-Servern erforderlich, die an späterer Stelle noch genauer beleuchtet werden. Einige Proxy-Server im SVN zeigten ein ungewöhnliches Verhalten im Fehlerfall: Obwohl die Authentifizierung des Clients am Proxy fehlschlägt, geben sie einen HTTP-Statuscode 200 („Alles okay“) zurück und teilen dann den eigentlichen Fehlerstatus erst im HTTP-Body in Form einer HTML-Seite mit. In so einem Fall kann das von den Sensoren verwendete Verfahren zum automatischen Finden der korrekten Authentifzierungsmethode fehlschlagen. Um dem in Zukunft entgegenwirken zu können, unterstützt die API nun den Endpunkt `/api/system/identify`, der unauthentifiziert abgerufen werden kann und immer den String „HoneySens“ zurückgibt, wenn der Server funktionstüchtig ist. Dieser „*predictible Endpoint*“ kann sowohl von den Sensoren zur Überprüfung ihrer Konnektivität als auch für den im Abschnitt über Docker Swarm beschriebenen Healthcheck genutzt werden.

Weitere serverseitige Neuerungen hinsichtlich der Multi-Plattformfähigkeit sind sonst im User-Interface zu finden. Die Weboberfläche präsentiert nun Downloads und Kurzanweisungen zur Installation übersichtlich nach dem Hinzufügen von Sensoren und auch nur dann, wenn Firmware für die betreffende Plattform auf dem Server registriert wurde.

Honeypot-Services Das in den früheren Projektberichten erarbeitete Konzept, dynamisch beliebige containerbasierte Honeypot-Services auf den Sensoren zu verteilen, wurde mit Hinblick auf die HoneySens-Version 1.0 sowohl server- als auch clientseitig finalisiert. Hierfür waren an der API und der Weboberfläche nur noch kleinere Korrekturen nötig. Honeypot-Services werden wie zuvor beschrieben als Docker-

Images in einer internen Docker-Registry abgelegt. Die Kommunikation seitens des Servers mit dieser Registry, die ebenfalls innerhalb eines Containers betrieben wird, fand bisher aus Kompatibilitätsgründen mit der Docker-API mit Hilfe eines innerhalb des Server-Containers betriebenen Docker-Daemons und unter Nutzung des Docker-CLI (`docker load`, `docker push`) statt. Damit dieser seine Arbeit verrichten kann, war es allerdings bisher nötig, den Server-Container im privilegierten Modus, bzw. in neueren Docker-Versionen mit der Capability `SYS_ADMIN` zu starten. Dies verursacht Schwierigkeiten beim Betrieb des Containers mit Orchestrierungssystemen wie Kubernetes, OpenShift oder Docker Swarm, die standardmäßig mit unprivilegierte Containern arbeiten. Als Alternative wurde daher nun das Tool *skopeo*[3] in die Serverumgebung integriert. Dieses erlaubt die Kommunikation mit der Registry-API ohne dabei auf privilegierte Rechte oder Capabilities angewiesen zu sein.

Zusätzlich zu den genannten Maßnahmen wurde sichergestellt, dass die Registry selbst, für die der HoneySens-Server als Proxy und somit auch authentifizierender Endpunkt agiert, von außerhalb (also den Sensoren oder Administratoren) lediglich lesbar und nicht modifizierbar ist. Eine Modifikation der Registry-Inhalte kann nun nur noch vom Server-Container aus mit Hilfe von skopeo erfolgen - aus Nutzersicht also nur über die gesicherte HoneySens-API.

Integration in Container-Orchestrierungssysteme Mit dem Aufstieg der containerbasierten Virtualisierung mit Docker zum De-Facto-Standard innerhalb der Industrie gewinnen auch die aus Unternehmersicht wichtigen Tools zur Abbildung vollständiger Geschäftsprozesse an Bedeutung. Das Management von Containern mit Hilfe von Orchestrierungssystemen wie *Kubernetes* oder *Docker Swarm* ist ein wichtiger Bestandteil solcher Prozessketten. Da ebenfalls innerhalb dieser Projektphase eine Erneuerung der Infrastruktur des Sächsischen Verwaltungsnetzes hin zum sogenannten *SVN 2.0* stattfand, in deren Zuge Docker Swarm als Basisplattform für alle in Containern betriebenen Services ausgewählt wurde, musste der HoneySens-Server ebenfalls an die damit verbundenen Anforderungen angepasst werden.

Ganz im Sinne der Twelve-Factor App[4] ist es erstrebenswert, beim Entwurf und Betrieb von Containern Code und Daten strikt zu trennen, ein Container selbst sollte daher effektiv *stateless* sein und dazu Daten nicht lokal, sondern auf extern ausgelagerten Volumes ablegen. Somit ist es möglich, Container ohne Datenverlust jederzeit beenden und auf Grundlage ihrer Volumes neu erzeugen zu können.

Die wesentlichen Neuerungen in diesem Kontext umfassen:

Logging auf die Standardausgabe : Einige im Server-Container betriebene Prozesse erzeugen kontinuierlich Logdaten, die typischerweise in einem Verzeichnis wie `/var/log` niedergeschrieben werden. Im Docker-Kontext ist es jedoch wünschenswert, dass alle Logdaten direkt auf die Standardausgabe der sie erzeugenden Prozesse geschrieben werden[5]. Um dieses vereinheitlichte Logging zu ermöglichen, wurden Apache, MySQL, Beanstalk usw. instruiert, ausschließlich `/dev/stdout` und `/dev/stderr` als Logging-Kanäle zu nutzen.

Docker Compose : Mit Hilfe eines sog. *Compose-Files* und dem zugehörigen CLI-Tool `docker-compose` kann definiert werden, wie ein Service, der im Betrieb aus einem oder mehreren Containern zusammengesetzt ist, betrieben werden soll (Start-Parameter, Volumes, Netzwerkanbindung usw). Eine

[3] `https://github.com/projectatomic/skopeo`
[4] `https://12factor.net/`
[5] vgl. `https://success.docker.com/article/logging-best-practices`

solche Servicedefinition wurde für den Server hinzugefügt, um den Start einer Serverumgebung massiv zu vereinfachen: es müssen nur noch die relevanten Parameter im Compose-File an die lokalen Bedingungen angepasst werden. Eine solche Definition erleichtert zudem auch den Betrieb des Servers unter Docker Swarm, da die meisten Parameter für den Swarm-Service direkt aus dem Compose-File abgeleitet werden können.

Healthcheck : Neuere Docker-Versionen unterstützen einen sogenannten Healthcheck. Das ist ein beliebiges Kommando, das ausgeführt werden kann, um den Status eines Containers (intakt/defekt) rudimentär beurteilen zu können. Als erster Indikator für Fehlverhalten im Rahmen eines Cluster-Healthmanagementsystems ist es hilfreich, wenn alle Services ein solches Kommando hinterlegen. Dies wurde auch für den HoneySens-Server umgesetzt, dessen Healthcheck den im vorigen Abschnitt beschriebenen neuen API-Endpunkt `/api/system/identify` nutzt, um das grundsätzliche Funktionieren der Webschnittstelle zu überprüfen.

Umgang mit leeren Volumes : Bei ersten Tests des Containers in einer Swarm-Umgebung stellte sich heraus, dass dieser an seine Umgebung eine Reihe von Anforderungen stellt, die bisher implizit vorhanden, aber nicht explizit als solche deklariert wurden. Hierzu zählte, dass Docker-Volumes nur dann korrekt verarbeitet wurden, wenn es sich entweder um sog. *Named Volumes* handelte oder diese bereits aus einer früheren Installation vorinitialisiert waren. Falls die Daten- oder Datenbank-Pfade jedoch mit einem leeren Volume versehen wurden, schlug der Start des Containers fehl. Entsprechende Modifikationen zum Anlegen von Volume-Templates beugen diesem Fall nun vor.

Authentifizierende TLS-Proxy-Server : Eine weitere unerwartete Besonderheit in der neuen Swarm-Infrastruktur war, dass TLS-Verbindungen nicht mehr direkt am HoneySens-Server beendet, sondern von einem vorgeschaltenen als vertrauenswürdig eingestuften Proxy behandelt werden. Da Sensoranfragen aber inzwischen gegenüber dem Server unter Zuhilfenahme von TLS-Clientzertifikaten authentifiziert wurden, war eine Kommunikation zwischen den Komponenten nicht mehr möglich. Nach Abwägung verschiedener Optionen wurde als Lösung gewählt, dem Server mit Hilfe einer Umgebungsvariable beim Start einen vertrauenswürdigen Proxyserver zu benennen, dessen übermittelte Zertifikatsparameter vom Server weiterverarbeitet werden. Dieser Proxy benötigt für den Betrieb allerdings das interne CA-Zertifikat der HoneySens-Installation, um Anfragen ordnungsgemäß überprüfen zu können.

Sonstiges Viele weitere Änderungen sind in die HoneySens-Version 1.0 eingeflossen, viele davon in Zusammenarbeit mit der T-Systems MMS. Um das Repository leichter zugänglich zu machen, wurden beispielsweise alle von der API benötigten externen PHP-Bibliotheken aus dem Repository entfernt und durch eine Abhängigkeitsverwaltung ersetzt (Composer[6]). Beim Bauen des Projektes mit dem *Grunt Task Manager* werden nun auch automatisch alle Abhängigkeiten der API aufgelöst und aus dem Composer-Repository heruntergeladen. Dieser Schritt verschlankte das Repository massiv, erleichtert aber auch zukünftige Updates einzelner Komponenten auf neuere verfügbare Versionen. Einige Frameworks wurden bei dieser Gelegenheit auch direkt aktualisiert, um sicherzustellen, dass die Komponenten des Servers selbst keine Sicherheitslücken oder kritischen Bugs beinhalten.

Weiterhin wurde das interne Zertifikatsmanagement des Servers überarbeitet. Falls ein Server mit selbstsignierten TLS-Zertifikaten betrieben wird, werden diese nun ebenfalls aus der internen HoneySens-CA

[6] `https://getcomposer.org/`

erzeugt. Diese dient natürlich auch weiterhin als CA für alle Sensor-Zertifikate. Eine separate Management-Schnittstelle, mit der die vergebenen Zertifikate besser verwaltet werden können, ist für die Zukunft geplant.

Die produktive HoneySens Version 1.0 enthält abgesehen von den zuvor genannten neuen Features unzählige weiter Detailverbesserungen bei der Installation, dem Updateprozess, dem Mailversand und bei der Darstellung auf Mobilgeräten. Weiterhin wurden inzwischen überflüssige und ungenutzte Code-Reste aufgeräumt (darunter ganze UI-Module) sowie die Endpunkte der REST-API konsolidiert, wenn sinnvoll. So ist beispielsweise `/api/sensorstatus/` nun keine eigene Ressource mehr, sondern wird vom Sensor-Endpunkt `/api/sensors/status` behandelt, was semantisch passender ist. Das Feedback der Betreiber der derzeit laufenden HoneySens-Systeme und die damit verbundenen Erfahrungen bei Installation, Betrieb und Updates waren hierbei ein wertvoller Betrag.

2.3 Veröffentlichung der Version 0.2.5

Parallel zu den zuvor genannten Arbeiten wurde die bisher produktive eingesetzte Version 0.2.4 des Servers weiter gepflegt, um ein OpenSource-Release dieses älteren Softwarestranges zu ermöglichen. Hierfür wurde u.a. das Basis-Image des Server-Containers auf die LTS-Variante von *Ubuntu 16.04* aktualisiert, bei Bedarf ein automatisches Updateverfahren für das MySQL-Tabellenformat angestoßen, viele kleinere Bugs behoben und ein Info-Modul ins Frontend integriert. Letzteres klärt über den Softwarestand, die Lizenz (*Apache-Lizenz 2.0* für die OSS-Variante) sowie die Autoren der Software auf (vgl. Abb. 3). Zudem wurden das Benutzer- und das Installationshandbuch als direkte Downloads integriert.

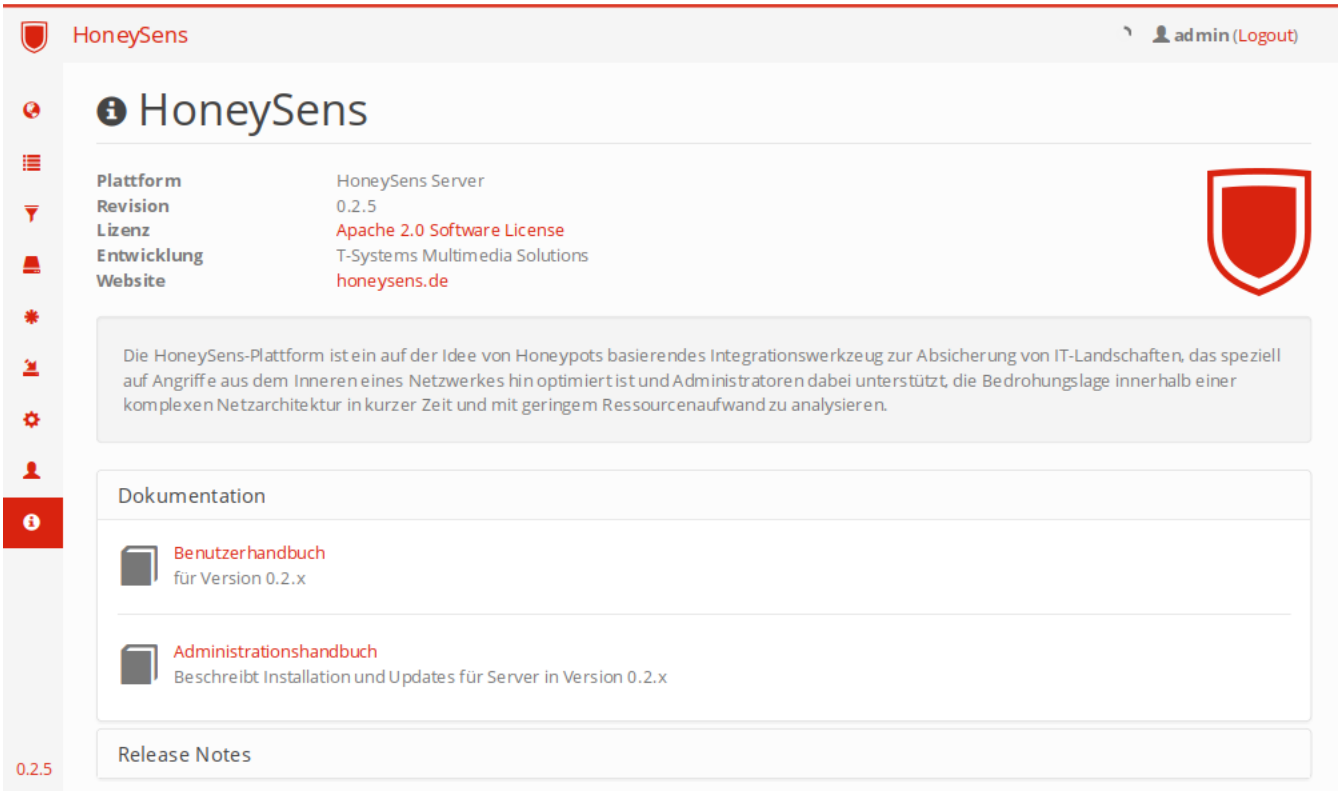

Abbildung 3: Informations-Modul im Frontend

Anschließend wurde die Version 0.2.5 auf GitHub[7] als Legacy-Variante veröffentlicht. Eine saubere Trennung zwischen der zukünftigen OpenSource-Variante und der darauf aufbauenden kommerziellen Version wird in einem davon getrennten Repository erfolgen.

Alle Änderungen der Version 0.2.5 sind nach deren Fertigstellung auch in den Entwicklungszweig der Version 1.0 eingeflossen und ergänzen deren Neuerungen.

2.4 Sensoren

Dieser Abschnitt gibt einen Überblick über weitere, noch nicht in vorherigen Berichten besprochenen Veränderungen in der Sensorsoftware im Kontext der HoneySens-Version 1.0. Dies umfasst sowohl allgemeine Veränderungen am Sensor-Manger, der mit der neuen Firmware erstmals zum Einsatz kommt und dessen Aufgabenfeld bereits zuvor klar abgesteckt wurde, als auch weitere plattformspezifische Anpassungen für die Sensoren auf Basis von Docker oder der BeagleBone-Hardware. Abschließend werden Änderungen an der nun containerbasierten Honeypot-Service-Infrastruktur beleuchtet.

Anpassungen des Sensor-Managers Der Management-Daemon, der auf allen Plattformen zum Einsatz kommt, ist u.a. für die Steuerung des Polling-Prozesses mit dem Server zuständig. Für die Authentifizierung bei diesem Vorgang gegenüber dem Server wurde bisher die zu übermittelnde Nachricht mit dem Sensor-Schlüssel signiert und von der Gegenseite, der die Sensor-Zertifikate vorliegen, überprüft. Mit Einführung der Service-Registry (siehe oben) war allerdings auch eine Authentifizierung der Sensoren an dieser erforderlich. Das offizielle Docker-Registry-Image, das als Bestandteil des Servers zum Einsatz kommt, unterstützt die bisher eingesetzten Signaturen als Teil der übermittelten JSON-Nachricht leider nicht. Alternativ ist es aber möglich, eine Authentifizierung mittels standardisierten TLS-Client-Zertifikaten durchzuführen. Aus diesem Grund wurde das bisherigen Verfahren zusätzlich um TLS-Client-Authentifizierung gegen alle für Sensoren relevanten API-Schnittstellen erweitert.

Weiterhin traten im Betrieb Probleme mit einer Reihe von Proxy-Servern auf, über die keine Kommunikation möglich war. Eine genauere Untersuchung des Problems hat ergeben, dass eine NTLM-Authentifizierung gegen Proxies auf Basis des *Microsoft Threat Management Gateways* (TMG) mit Hilfe der weit verbreiteten Bibliothek *libcurl* fehlschlug. Da viele Proxies im SVN aber genau diese Form der Authentifizierung verlangen, wurde die Sensorsoftware um das Tool *cntlm*[8] erweitert. Es öffnet einen lokalen Netzwerksocket und sendet alle Pakete, die es erhält, über zuvor konfigurierte Proxy-Server weiter. NTLM-Authentifizierung funktioniert mit dieser Software problemlos, wodurch die Anbindung der Sensoren im Landesnetz wiederhergestellt werden konnte.

In einem letzten Schritt wurden die von den Sensoren während des Pollings an den Server übermittelten Statusdaten um Informationen über den verbleibenden freien Plattenspeicher erweitert. Hintergrund ist, dass jeder auf einem Sensor betriebene Honeypot-Dienst zunächst als Docker-Image vom Server heruntergeladen und lokal gespeichert werden muss. Mit der wachsenden Zahl an verfügbaren Honeypot-Services kann es vorkommen, dass dieses Vorgehen die Speicherkapazität des Sensors erschöpft. Insbesondere die BeagleBone-Boards nutzen lediglich einen internen 4 GB großen Flash-Speicher, der nur

[7] `https://github.com/sylencecc/HoneySens-Legacy`
[8] `http://cntlm.sourceforge.net/`

eine begrenzte Menge an parallel betriebenen Services erlaubt. Um nachvollziehen zu können, ob noch Platz auf einem Sensor für weitere Services zur Verfügung steht, wurde die Auswertung dieser neuen Statusdaten in die Übersicht der Sensoreigenschaften des Frontends integriert. In der Zukunft ist eine graphische Auswertung dieser Werte angedacht.

Betrieb von Docker-Sensoren Sensoren können nun dank der Plattformunabhängigkeit des Management-Daemons auch als Docker-Container betrieben werden. Auch hierfür wurde in der vergangenen Projektphase eine prinzipiell funktionstüchtige Grundlage gelegt, die im aktuellen Sprint noch ausführlich getestet und um währenddessen aufgetretene Bugs bereinigt wurde. Ein Umdenken erforderte die Anbindung eines solchen Sensors ans Netzwerk: Der Plan sah zunächst vor, eine Netzwerkschnittstelle des Hostsystems exklusiv an den Sensor-Container weiterzureichen und dann wie auch bei den BeagleBones über das Web-Interface zu konfigurieren. Der Nachteil dieser Methode ist, dass pro virtuellem Sensor zwingend ein separates Interface mit eigener IP-Adresse erforderlich ist, was insbesondere beim Einsatz in Unternehmen oftmals unnötigen Overhead darstellt. Auch der Versuch, ein bereits vom Hostsystem konfiguriertes Netzwerk-Interface mitzunutzen, indem der Container ebenfalls auf dem Netzwerkstack des Hostsystems arbeitet, führten lediglich zu Problemen und unzähligen False-Positives durch vom Host verursachten Datenverkehr. Die schließlich implementierte Lösung vermeidet Raw-Sockets und exklusiven Zugriff auf Interfaces und setzt stattdessen auf die vom Docker-Projekt standardmäßig zur Verfügung gestellten Möglichkeiten, Container zu vernetzen: Ein Sensor-Container ist einem sog. *„Docker User Defined Network“* zugewiesen und nur über eine vom Hostsystem aus ansprechbare, interne IP-Adresse erreichbar. Es ist dann Aufgabe des Administrators, selektiv einzelne TCP/UDP-Ports oder auch jeglichen eingehenden Netzwerkverkehr an den Container mittels *Netfilter* weiterzuleiten. Der Container ist schließlich für die Filterung und Weiterleitung der Pakete an die zuständigen Honeypot-Service-Container verantwortlich (Details dazu im nachfolgenden Abschnitt). Dieses Verfahren hat sich in der Praxis bewährt und gewährt Betreibern die nötige Flexibilität, Sensor-Container innerhalb beliebiger virtueller Maschinen zu betreiben, die nun bei Bedarf auch noch für andere Aufgaben oder Container genutzt werden können. Nachfolgend exemplarisch zwei Netfilter-Regeln, mit denen aller eingehender Traffic, der vom Kernel als neuer Verbindungsaufbau klassifiziert wird, an den Sensor-Container weitergeleitet wird:

```
$ iptables -t nat -I PREROUTING -i <interface> -m state \
  --state RELATED,ESTABLISHED -j ACCEPT
$ iptables -t nat -A PREROUTING -i <interface> -j DNAT \
  --to-destination <sensor-IP>
```

Weiterhin wurde der Docker-Firmware (die letztendlich ein gepacktes Docker-Image ist) ein Launcher-Skript hinzugefügt, das Administratoren zum leichten Verwalten von Docker-Sensoren einsetzen können. Es vereinfacht den Start neuer Sensor-Container mit allen nötigen Parametern und führt automatisch ein Update des Sensor-Images aus, falls davon eine neue Revision vorliegt. Hierfür sollte das Skript regelmäßig im Update-Modus beispielsweise von *cron* ausgeführt werden. Es sucht dann in einem Verzeichnis, das es sich mit dem Sensor-Container als Volume teilt, nach neuer Firmware, importiert diese bei Bedarf, beendet den bisherigen Sensor-Container und startet eine neue Instanz auf der Grundlage der neuen Firmware.

Firmware für den BeagleBone Black Der Einplatinencomputer BeagleBone Black ist die älteste von HoneySens unterstütze Hardware-Plattform. Für die zugehörige Firmware wurde im Rahmen dieser Projektphase ebenfalls ein Buildprozess entworfen, der auf dem offiziellen Repository des BeagleBone-Image-Builders[9] fußt und über ein Makefile zugänglich ist. Das Erstellen der Firmware muss jedoch zwingend auf einem System mit ARM-Architektur erfolgen, QEMU wird nicht unterstützt.

Im gleichen Zuge wurden einige plattformspezifische Besonderheiten zum Sensor-Manager hinzugefügt. So stellt eine neue Firmware-Updateprozedur sicher, dass vor einem Update laufende Honeypot-Services zunächst vollständig entfernt werden, um möglichst viel Plattenplatz für das neue Image freizugeben. Außerdem wurde der Code zur Ansteuerung der LED-Erweiterungsplatinen, mit denen viele Sensoren im SVN ausgestattet sind und die ein visuelles Feedback über den Sensorzustand geben, überarbeitet.

Honeypot-Services Die Honeypot-Funktionalität der Sensoren wird beginnend mit HoneySens 1.0 durch in Docker-Containern betriebene OpenSource-Services wie beispielsweise *cowrie* (SSH-Honeypot) oder *Glastopf* (HTTP) abgebildet. Diese Dienste können dynamisch von Nutzern auf dem Server registriert und frei auf die Sensoren verteilt werden. Für dieses Vorgehen galt es, verschiedene zusätzliche Anforderungen zu erfüllen. So war es beispielsweise erforderlich, Service-Images für verschiedene Hardware-Architekturen bereitzustellen, da BeagleBones ARM-CPUs nutzen, die Docker-Container aber für x86-Systeme vorgesehen sind. Docker-Images für verschiedene Architekturen sind auf dem Docker-Hub bereits verfügbar und es ist auch möglich, die Zielarchitektur als Teil der Metadaten eines Docker-Images selbst festzulegen. Die Metadaten der in HoneySens genutzten Services wurden daher ebenfalls um diese Information angereichert und die serverseitige API sowie das Webfrontend entsprechend angepasst. Es ist nun für Nutzer schnell ersichtlich, für welche Architekturen ein bestimmter Service zur Verfügung steht. Die Sensorsoftware stellt zudem sicher, dass nur Docker-Images vom Server bezogen werden, wenn die jeweilige lokale Architektur auch unterstützt wird.

Beim Konvertieren frei verfügbarer Honeypot-Software in das für HoneySens genutzte Container-Format haben Entwickler jetzt außerdem die Möglichkeit, die Netzwerkanbindung des Service-Containers über die Metadaten (die im XML-Format zusammen mit dem Service-Abbild verpackt werden) festzulegen. Folgende Möglichkeiten stehen zur Verfügung:

Gezielte Portzuweisungen : Mit der Syntax `{„2222/tcp":22}` können ein oder mehrere interne Ports des Sensor-Containers nach außen hin geöffnet werden. Im Beispiel würde der Dienst, der innerhalb des Containers auf dem TCP-Port 2222 läuft, von außen über den TCP-Port 22 erreichbar sein. Somit können Container auch Dienste auf privilegierten Ports anbieten, ohne gesonderte Rechte zu benötigen. Die Weiterleitung selbst wird vom Sensor-Manager gesteuert.

Catch-All-Dienste : Wenn ein Dienst das Flag `catchAll` aktiviert, wird bei dessen Start vom Sensor-Manager eine Netfilter-Regel angelegt, die jeglichen Traffic, der nicht bereits von einem der anderen Dienste verarbeitet wurde, an den neuen Dienst weiterleitet. Mit dieser Funktionalität wurde der zuvor eigenständige *recon*-Dienst, der Verbindungsversuche via TCP und UDP auf beliebigen Ports aufzeichnen kann, in die neue Architektur integriert.

Raw-Zugriff : Ein Honeypot-Service erhält durch Aktivieren dieses Modus Raw-Zugriff auf das zugrundeliegende Netzwerkinterface des Sensors, ist jedoch selbst für die Filterung des sichtbaren

[9] `https://github.com/RobertCNelson/omap-image-builder`

Datenverkehrs verantwortlich. Derzeit nutzt kein Dienst diese Funktion, eine spätere Integration von `honeyd` oder `arpd` zur Simulation zusätzlicher Honeypot-Endpunkte mit nur einem Netzwerkinterface oder die Integration von Honeypots, die nicht auf dem IP-Protokoll basieren, ist hiermit denkbar.

Die Auswahl an Honeypot-Diensten wächst zudem beständig: Neben den bisher von der alten Architektur erfolgreich konvertierten Projekten *cowrie* (Nachfolger von kippo) und *recon* stehen inzwischen auch noch Glastopf (Web-Application-Honeypot), RDPy (simuliert Remote-Desktop-Verbindungen) und der Conpot zur rudimentären Simulation von Industriesteueranlagen (ICS) zur Verfügung.

3 Ausblick

In der sich anschließenden Projektphase soll der Fokus auf der besseren Auswertung und dem Verständnis der von den Sensoren erzeugten Ereignissen liegen. Aus dem Praxisbetrieb gewannen wir die Erkenntnis, dass die Auswertung der gesammelten Ereignisdaten mit zunehmender Menge Schwierigkeiten bereitet. Die Ereignisliste reicht dann als alleiniges Mittel zur Verarbeitung der Daten nicht mehr aus und es gilt, darüber nachzudenken, wie die Daten sinnvoller aggregiert und dem Nutzer visualisiert werden können, so dass nur noch für Detailfragen auf die Ereignisliste zurückgegriffen werden muss. Weiterhin ist auch eine Anbindung an SIEM-Systeme in Betracht zu ziehen, die Honeypot-Daten mit Informationen aus anderen Quellen korrelieren können.

Als Grundlage für die nächsten Schritte in diesem Feld dient ein Datensatz aus dem Testbetrieb an der TU Dresden über die vergangenen Jahre, der zunächst anonymisiert und dann nach Anomalien und Auffälligkeiten hin untersucht werden soll. Das Ziel wird sein, geeignete Verfahren zur zumindest halbautomatischen Auswertung und besseren Visualisierung der gesammelten Daten zu finden, auf den Datensatz anzuwenden und geeignete Algorithmen auch in den Server zu integrieren.

Weiter in die Zukunft gedacht, ergeben sich für die Weiterführung des Forschungsprojektes eine ganze Reihe von interessanten Ansatzpunkten:

Erweiterung der Honeypot-Funktionalität : Mit Fertigstellung der modularen Honeypot-Services existiert nun ein standardisiertes Containerformat, in dem leicht neue OpenSource-Honeypot-Projekte in HoneySens eingebunden werden können. Hier gilt es, eine sinnvolle Vorauswahl zu treffen und in produktiven Netzen häufig anzutreffende Dienste abzubilden.

Sichtbarkeit : Die Betrachtung dieser Thematik aus wissenschaftlicher Sicht wurde bereits in Zwischenberichten erörtert. Als ein unmittelbarer Schritt wäre allerdings denkbar, Sensoren mit mehreren fixen IP-Adressen gleichzeitig auszustatten, um Administratoren ein Werkzeug zur Verfügung zu stellen, mit dem einzelne Sensoren mit statischen Mitteln für potentielle Angreifer prominenter platziert werden können. Diese Lösung könnte insbesondere für KMUs interessant sein, um mit nur wenigen Sensoren eine größere Bandbreite des Unternehmensnetzes abdecken zu können.

Firmware-Signatursystem : Um dem Fall vorzubeugen, dass ein erfolgreicher Angriff gegen den Server in manipulierter Firmware auf den Sensoren resultiert, ist ein Verfahren zu entwickeln, mit dem Firmware bei ihrer Erzeugung signiert und vor der Installation automatisch überprüft werden kann.

4 Anhang

Diese Auflistung soll die zuvor beschriebenen Prozesse und Veränderungen anhand der im Laufe des Forschungsprojektes in der Versionsverwaltung hinterlegten Kommentare für jede neue Softwarerevision dokumentieren.

Revision	Änderungen
462	- Platforms moved into the database and served as a separate API resource - Platform frontend module added - Update worker prepared for updates to version 1.0.0
463	Branch path added
464	Preparations for MMS 1.0
465	Fixed a bug that prevented system-wide settings to be validated correctly when no SMTP server was configured
466	Fixed a bug that prevented managers from changing the status flag of events
467	Server: Improved error handling for the setup module
468	Server: Info module added to the UI with full documentation download, release notes and generic project information
469	User and admin manual sources
470	- Provisional 1.0.0 version to trigger db updates Backend: - Firmware management migrated into 'platforms' resource - Sensor configurations removed as separate entity - Sensors now also keep update interval and prefered firmware rev. - BBB and Docker backend platforms added Frontend: - Platform module handles firmware management - Sensor interval configurable during sensor configuration
471	- Update interval is now properly updated when configured in the frontend - Database update script to 1.0.0 simplified
472	Donut menu is now properly displayed and stylized
473	Server update script restructured to perform all SQL updates line-by-line regardless of errors
474	Differential update DB table now properly updates with the 1.0.0 version step

475	- Specific firmware revisions for individual sensors can now be configured as port of the regular sensor configuration dialog - Sensor update interval moved to the global settings module (and to the server configuration file)
476	- All references to sensor configs and sensor images removed from both frontend and backend - Various other code cleanups, especially deprecated UI modules - Code that was based on sensor configs rewritten to point to the new location of the former values (mostly directly to sensors)
477	- Sensor certificate checks introduced for sensor-related endpoints - Sensorstatus API resource merged into Sensor resource - UI: Fixed a bug that prevented new sensors from being added
478	Server: - Certificate management overhaul: self-signed TLS certs are now signed using the internal CA - Docker firmware handling implemented (stored in the local registry) - UI: Sensor timeout display bug fixed Sensor: - Client certificate auth support - 'generic' platform renamed to 'docker_x86' - Launcher script added to start and update docker sensors - Busybox 'ifup' wrapper added for python-debinterface compatibility
479	Proxie'd docker registry headers added for proper operation
480	Server now properly removes firmware from the registry
481	armhf build files for the cowrie service added
484	- Services are now differentiated by architecture (frontend and backend) - Services registry now uses the 'services' repository for services UI: - Platforms and services now properly auto-update - Service handling improved for upload, display and removal - Backgrid subcell extension integrated for nested grids
485	- Service manager picks services based on the local platform architecture - Docker platform launcher script updated for proper re-execution upon existing containers
488	Server version 0.2.5: - Baseimage adjusted and upgraded to Ubuntu 16.04 TLS - Generic MySQL update on launch script added (to upgrade table metadata if required) - Info module documentation URLs fixed - Info module now references T-Systems MMS as author

489	Cowrie service module reimplemented as an output plugin for cowrie 1.2.0, as well as Makefile and multiarch support added
490	BeagleBone sensor firmware generation scripts updated for reproducible builds
491	Sensor manager: Fixed a bug that caused the update interval to be fixed at 1 minute
492	- Fixed a bug that prevented the periodic web UI updates - Missing PHP-XML module dependency added - BBB Sensor firmware version updated to 0.2.4
493	Merge of the latest changesets of the the 0.2.5 branch, lots of bugfixes related to the merging as well as some repository restructuring, cleanups and some documentation
494	- Registry worker added to server container - MySQL update script adjusted for smooth startup - Doc updates: –privileged currently needed for server during 'docker load' - Cowrie service Makefile paths corrected
495	SSH disabled on the server container
496	Firmware selector when creating new sensors now defaults to „system default“
497	Certificate CN extractor routine updated to handle certificate chains
498	All external PHP dependencies removed from the repository and replaced by a composer dependency definition file, including a few naming tweaks
499	Composer automation via Grunt
500	Adjustment for compatibility with FileUpload v1.4.5
501	- Port assignments support for services added (has now to be specified in service definitions) - Raw network access for service containers is now supported - BBB firmware doesn't expose SSH anymore on eth0
502	Firmware metadata added
503	Legit SSH access fixed for Ethernet-over-USB (BBB)
504	Recon adjusted to pluggable service architecture and converted to a service
505	Manager: Spawn service containers with administrative network capabilities

506	- Option to leave a sensor network unconfigured added - Firmware download buttons added to the platform UI - Docker firmware naming fixed - Docker firmware is now stored in the filesystem instead of the registry (to streamline deployment and update of docker sensors) - API endpoint added that downloads the current firmware for a given platform
507	Manager: - CatchAll mode added for services (not yet activated) - Now accepts an interface parameter - Can leave an interface unconfigured if instructed to do so
508	Manager: Service catch-all mode properly implemented
509	Service revision metadata interpretation during upload fixed
510	Firmware update process implemented for docker sensors (both manager and launcher)
511	Added attach option to the docker launcher
512	Sensor manager: - Platform can now be selected via a command line argument - BBB module rewritten to support updates and proxies - Docker module error handling improved BBB platform: - Directory tree cleanup - Makefile added, build process streamlined Docker platform: - Makefile added, build process streamlined
513	BBB Makefile fixed to properly include the image size as part of the revision
514	- Fixed a bug where the manager skipped the „wait for interface“ routine by default - Revision marker added to BBB image
515	BBB Makefile fix
516	Rotated header cells for the service grid in the sensor overview
517	Disable the service UI as long as the registry isn’t online
518	- Sensor manager now also removes images together with containers to save space - BBB Platform: APT cache cleanup after installation added
519	Service UI: Table position now automatically adjusts depending on the service labels
520	Fixed a display bug of the service matrix with no services registered
521	Fixed a bug that prevented the activation of multiple services on a single sensor

522	UI: - Set uploaded firmware as default for each platform if there isn't a default yet - When no firmware is registered system-wide, sensors can now still be edited or created, but a respective warning is shown and the firmware selector is disabled - Bootstrap updated to 3.3.7
523	API: - Default firmware revision can't be removed - Sensors configured to target a specific firmware revision are reset to default in case that revision is removed UI: - Textual warning of the aforementioned behaviour prior to removal
524	- Sensor manager version set to 1.0.0 BBB platform: - SSH over USB fixed, didn't launch in a few cases - SSH default credentials added (public key) - LED handling - More robust and less resource-intensive updates
525	Sensor installation instructions simplified and only displayed for available firmware
526	- BBB updates: Remove existing containers and images to save space - Sensor manager: Proxy support now provided by cntlm, which means exclusive NTLM support - Launcher script is now part of the Docker firmware - Fixed a bug concerning the calculation of service assignments - Predictable API endpoint added for proxy checks: /api/system/identify - Fixed a UI bug where sensor installation instructions were sometimes not visible - Server version switched to 0.9.0
527	Fixed a bug where under certain conditions service uploads failed
528	Launcher bug fixed to specify a specific network
529	BBB firmware: LED control script made executable
530	Password confirmation field added when editing users
531	UI: API endpoint corrected for fetching sensor status data
532	UI: MMS branding target
533	- Beanstalk services now run as user 'beanstalkd' - Apache and MySQL logging redirected to /dev/stdout - Docker compose file added - Server healthcheck added - Makefile config option for branded builds added

534	Removed a few UI controls that were visible for users of the group „Beobachter“, but shouldn’t be
535	Glastopf service added
536	Compose file comments added that indicate how to supply a separate TLS keypair
537	- BBB: Internal debugging SSH (over USB) daemon moved to port 22222 to avoid conflicts with docker-proxy Server: - Internal docker daemon for load/push replaced with skopeo, thus eliminating the need for SYS_ADMIN capabilities - Beanstalk workers now write to stdout unbuffered for logging consistency - Gruntfile release target fixed (was building with branding even though it shouldn’t)
538	API: Services can now be removed if the registry is online, but the given image isn’t stored there
539	Fixed an API bug that caused multiple services to be deactived when only a single one was deselected
540	API: - Proxy user and password handling made more consistent - Unused ’firmware’ entry removed from generated sensor configs - Now unused SYS_ADMIN capability removed from compose file
541	API: Client vertification disabled for /api/sensors/status/by-sensor/<id> to enable access from the UI
542	Server version bumped to 1.0.0 API: - Disk usage/total reporting from sensors added - Proxy user is now an optional argument when submitting sensors (fixes UI inconsistency) - Access to /api/sensors/status/by-sensors/<id> reenabled for the UI via apache config
543	- Server update process to 1.0.0 streamlined: old firmware is now removed automatically and the order of SQL statements was fixed - Server CA regeneration script now recognizes older server configurations and import those CAs
544	Server: - Can now re-initialize empty (data and db) volumes on startup - Implicit import of HTTPS certificates from server < 0.9.0 after upgrades is now supported - External registry access is now read-only (HEAD and GET methods)

545	Sensor manager: - Now properly rewrites the cntlm config file in case no authentication was configured - Configuration updater added to convert pre-0.9.0 configs to the current format (after firmware updates) - Polling now also reports disk usage stats - Grace timeout period and exception handlers added after containers are stopped to allow for a proper shutdown - Already registered containers on the system are now registered on startup and synchronized during the first successful polling - BBB platform issues a daemon-reload in case files under /etc/systemd changed BBB platform: - macchanger added to setup script
546	Sensor platforms BBB and docker version numbers set to 1.0.1 to incorporate the latest changes. Server 1.0.0 requires at least this version to be able to talk to sensors.
547	Server: - CA regeneration script will now issue a new CA cert and fix some path issues if a CA from server < 0.9.0 was found - TLS regeneration script is now aware of custom certs mounted into the container BBB firmware: - Build phase made noninteractive again (macchanger was prompting for stuff)
548	BBB Firmware version 1.0.2: - dnsmasq that provided DHCP via USB disabled to close ports it opened on 0.0.0.0 - resolvconf installed for proper static DNS nameserver handling
549	Still BBB firmware 1.0.2: Fixed a bug that prevented a few packets from being installed during the build process
550	Still server 1.0.0: Support for remote proxies that perform TLS client authentication
551	- Vagrantfile added that creates a uniform dev server environment - A couple of improvements for the docker setup script
552	Build and init scripts rewritten: Now everything can be built or developed inside the docker-dev container/image
553	- Vagrantfile added to support the dev environment within a VM - Re-enabled grunt-simple-watch - Dev notification added
554	Various scripts made executable
555	Boot2Docker Vagrantfile
556	RDPy v1.3.2 service added