Technische Universität Dresden

Fakultät Informatik

Professur für Datenschutz und Datensicherheit

Projekt HoneySens

2. Zwischenbericht

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

Stand: 16.10.2015

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. Ein weiterer wichtiger Bestandteil des Forschungsprojektes ist zudem die Koordinierung der vielfältigen Anforderungen der am Projekt interessierten und teilnehmenden Ressorts der Landesverwaltung.

Während die Ziele des ersten Zwischenberichtes insbesondere auf die Stabilisierung der Plattform und die zukünftige Erweiterbarkeit des Projektes durch ein umfangreiches Refactoring ausgerichtet waren, beinhaltete die zweite Iteration eine Reihe umfangreicher Erweiterungen, die sowohl die Qualität der vom System gesammelten Daten, als auch dessen Flexibilität deutlich verbessern. Hierzu zählen insbesondere ein serverseitiger Filtermechanismus für eintreffende Ereignisse, ein neuer Build-Prozess für die Sensorsoftware im Sinne der Plattformunabhängigkeit, sowie ein neuer interaktiver Sensordienst, der ausführlichere Informationen über eintreffende Pakete gewinnen kann als der bisher verwendete „Passive Scan Mode“. Diese Punkte wurden allesamt im „Ausblick“ des vorigen Berichtes angesprochen und in dieser Iterationsstufe auch erfolgreich umgesetzt. Die Verbesserungen und Neuerungen sind zudem weiterhin auf einen zukünftigen, möglichst reibungslosen Testbetrieb innerhalb des Sächsischen Verwaltungsnetzes ausgerichtet, der nun als Teil der dritten Iterationsstufe beginnen soll. Das Beheben neu entdeckter Fehler und die konstante Verbesserung von Buildprozess und Projektstruktur in kleinen, iterativen Schritten (*Refactoring*) wurden analog zur ersten Iterationsstufe fortgeführt und werden in diesem Dokument nicht im Detail erklärt. Die Commit-History aus der Versionskontrolle im Anhang soll als Referenz dienen.

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.

2 Sachstand

Dieser Abschnitt gibt einen Überblick über die vom 16. Juli bis zum 16. Oktober 2015 durchgeführten Arbeiten am HoneySens-Projekt. Teilbereiche, die in den Anforderungen des ersten Projektberichtes genannt, aber hier nicht weiter beschrieben werden, sind Gegenstand der zukünftigen Weiterentwicklung.

Systemwartung Eine insbesondere während des Testens und Evaluierens der Anwendung aufgetretene Anforderung ist der Wunsch nach leicht erreichbaren Funktionen zum Zurücksetzen des Systems in den Auslieferungszustand und die Möglichkeit, alle bisher gesammelten Ereignisse zu entfernen. Das ist insbesondere dann hilfreich, wenn registrierte Sensoren zu Testzwecken große Mengen an Daten (bei-

spielsweise wiederholte Portscans) erhalten und die Ereignisliste als Resultat beginnt, unübersichtlich zu werden. Es kam zudem während der Entwicklung und auch während eines Praxistests noch bei der Evaluation der ursprünglichen Diplomarbeit zu Situationen, in denen Fehler in den Sensordiensten dazu führten, dass tausende individuelle Ereignisse auf einen Schlag registriert wurden. Um nach einem solchen Vorfall die Ereignisliste wieder in einen nutzbaren Zustand versetzen zu können, wurden für Administratoren über das Web-Frontend erreichbare Funktionen zum Entfernen aller Ereignisse und zum vollständigen Zurücksetzen der Datenbank in den Anfangszustand integriert. Dies beinhaltet ebenso das Anlegen eines administrativen Standardnutzers.

Plattformunabhängigkeit Nachdem der Versuch in der ersten Iteration, BeagleBones mit Hilfe von *Qemu* zu emulieren, aufgrund von unzureichender Unterstützung für diese Geräteplattform (und ihrer Besonderheiten) aufgegeben werden musste, wurde im Rahmen der zweiten Projektphase nun ein anderer Ansatz verfolgt. Zunächst wurde die Sensorsoftware aufgeteilt, sodass die plattformspezifischen BeagleBone-Komponenten von den generischen Programmteilen getrennt waren. Anschließend war es möglich, den generischen Teil um *Skripte zur Paketerzeugung* für Debian GNU/Linux zu ergänzen. Hierfür wurden die Richtlinien des *Debian New Maintainers' Guide* zur standardkonformen Erzeugung von Softwarepaketen berücksichtigt. Es ist somit nun mit nur geringem Aufwand möglich, Debian-Pakete mit der Sensorsoftware für verschiedene Hardwarearchitekturen (darunter *x86_64* und *ARM*) zu erzeugen und wie jede andere Software auch auf Zielsystemen zu installieren. Dies ist ein erster wichtiger Schritt, um die Auswahl der als Sensoren in Frage kommenden Geräte zu erhöhen. So wurden beispielsweise für alle durchgeführten Versuche und Weiterentwicklungen seit Erzeugung dieser Pakete x86-basierte virtuelle Maschinen als Sensoren verwendet, was in Zukunft ebenso bei der Simulation großer IT-Infrastrukturen (beispielsweise für Belastungstest) hilfreich sein kann. Auch auf BeagleBones können die Pakete installiert werden, das Erzeugen vollständiger Abbilder für SD-Karten, die ohne Umwege direkt von den Boards verwendet werden können, ist jedoch erst Bestandteil der nächsten Projektphase. Der jetzige Schritt war hierfür aber bereits eine wichtige Voraussetzung.

Im Rahmen der Isolierung, Aufteilung und Aktualisierung der Sensorsoftware wurde diese zudem auch an einigen Stellen modernisiert. So lesen beispielsweise die Python-Module jetzt immer die systemweite HoneySens-Konfigurationsdatei aus. Zuvor fand dieser Prozess nur bei der Erstinstallation des Sensors statt, was eine spätere Veränderung der Parameter verkomplizierte.

Whitelisting Einer der wichtigsten Eckpunkte in der Liste der Anforderungen waren Filtermöglichkeiten, um die Flut an eingehenden Ereignissen kontrollieren zu können. Ein derartiger Ansatz zum sog. *Whitelisting* von Ereignissen wurde untersucht und implementiert. Er erlaubt es, am Server registrierte Ereignisse auf verschiedene Kriterien hin zu untersuchen und bei Übereinstimmung mit hinterlegten Filterregeln automatisch zu verwerfen. Derartige Filterregeln berücksichtigen derzeit Kriterien wie die Klassifikation eines Ereignisses, die IP-Adresse des Quell-Hosts (wahlweise einzeln oder als Adressbereich), der am Sensor angesprochene Ziel-Port oder das verwendete Layer-4-Protokoll (aktuell *TCP* oder *UDP)*. Diese Kriterien können beliebig miteinander kombiniert und zu einem auf dem Server registrierten Filter zusammengesetzt werden, der anschließend zudem mitzählt, wie viele die Kriterien erfüllende Ereignisse bisher aufgetreten sind, was sich beim Testen der Filterregln als ausgesprochen hilfreich herausstellte. Die graphische Benutzeroberfläche zur Verwaltung der Filterregeln orientiert sich an der Filterverwaltung, wie sie häufig auch in E-Mail-Clients verwendet wird und sollte deshalb für Administratoren des Systems schnell zu erlernen sein. Weiterhin wurden bei der Implementierung die Anforderungen der

Mandantenfähigkeit berücksichtigt, das heißt insbesondere die Zuordnung der Filter zu jeweils einer frei wählbaren Benutzergruppe und die Respektierung dieser Wahl bei der späteren Anwendung der Regeln.

Im praktischen Einsatz ist das Whitelisting immer dann nützlich, wenn gehäuft unkritische Ereignisse an Sensoren registriert werden, die sich angesichts der Netzwerkstruktur nicht vermeiden lassen. Dies könnten beispielsweise automatisierte, regelmäßige Scans aller ungenutzten IP-Adressen zum Zwecke der IT-Sicherheit sein. In einem solchen Fall kann das scannende System mit Hilfe einer Filterregel als harmlos deklariert und in Zukunft ignoriert werden.

Recon-Service Eine weitere, ebenso aus Sicht der Forschung interessante Anforderung war das Finden und Implementieren einer Möglichkeit, sensorseitig unter kontrollierten Bedingungen beliebige TCP-Verbindungen anzunehmen, um das erste Datenpaket eines potentiellen Angreifers aufzuzeichnen. Ebenso ist auch der Datenteil von verbindungslosen UDP-Paketen interessant, um bei einer späteren Analyse Rückschlüsse auf die Natur eines versuchten Angriffes ziehen zu können. Aus diesem Grund wurde der *Passive Scan Mode*, ein Python-Skript, das mit Hilfe des kerneleigenen *Netfilter*-Frameworks, *rsyslog* und `iptbales` bisher lediglich Verbindungsversuche ohne Datenanteil aufzeichnete, durch einen komplett neuen Dienst ersetzt, der *Recon-Service* (bzw. Reconnaissance Service) genannt wurde. Dieser sollte die Kernfeatures seines Vorgängers, darunter die integrierte Erkennung von Scanversuchen, beibehalten und möglichst performant um die Fähigkeit ergänzen, auf TCP-Verbindungsanfragen aktiv zu antworten, ein eventuelles erstes Datenpaket aufzuzeichnen und die Verbindung wieder protokollkonform zu schließen.

Bei der Analyse der Sachlage wurden verschiedene Lösungsmöglichkeiten in Betracht gezogen, darunter auch die Implementierung eines Dienstes, der jeweils einen TCP-Port und einen UDP-Port geöffnet hält, das beschriebene Antwortverhalten implementiert und alle interessanten Pakete beliebiger Ports beispielsweise über `iptables`-Regeln an seinen lokalen Port weitergeleitet bekommt. Dieses Verfahren besitzt jedoch neben der Abhängigkeit vom *Netfilter*-Framework noch weitere Nachteile, darunter eine erschwerte Flusskontrolle im Falle von Portscans oder versuchten *Denial-of-Service-Angriffen* (DoS) auf den Sensor. Letztendlich fiel die Wahl auf die Nutzung eines vom Betriebssystem bereitgestellten *Raw-Sockets*, mit dessen Hilfe Pakete am TCP/IP-Stack vorbei empfangen und versendet werden können. Es ist dabei insbesondere wichtig, sicherzustellen, dass gewollte Kommunikation zwischen externen Hosts und lokal laufenden Honeypot-Diensten wie *kippo* und *dionaea* nicht durch den neuen Dienst kompromittiert oder blockiert wird.

Das Python-Framework *scapy* stellte sich als eine ideale Grundlage für die Implementierung heraus, da es dem Entwickler den Zugriff auf Raw-Sockets auf sehr komfortable Art und Weise zur Verfügung stellt und das Erstellen von neuen und Verarbeiten von eintreffenden Paketen massiv erleichtert. Netzwerkpakete können mit Hilfe von scapy komfortabel schichtweise (IP-Layer, TCP-Layer usw.) zusammengesetzt werden, wobei ausgelassene Schichten automatisch ergänzt und sinngemäß parametrisiert werden. Das Resultat der Arbeiten ist eine Python-Anwendung, die folgende Funktionen übernimmt und damit den Prozess, Inhalte von eingehenden TCP- und UDP-Paketen aufzuzeichnen, vollständig abbildet:

- Analyse aller auf dem Host laufenden Dienste und Anlegen entsprechender Ausnahmeregeln, um deren Betrieb nicht zu beeinträchtigen
- Aufzeichnen und ggf. Beantworten aller Pakete, die direkt an die IP-Adresse des Hosts gerichtet und nicht für einen anderen Dienst bestimmt sind
- Nach Bedarf Durchführung von TCP-Handshakes und ordnungsgemäßes Beenden von TCP-Verbindungen

- Sammeln aller von einem potentiellen Angreifer eintreffenden Pakete innerhalb eines fixen Zeitraumes x und Zusammenfassung dieser als einzelnes Ereignis
- Senden der Ereignisse zum HoneySens-Server
- Flusskontrolle zur Vermeidung von DoS-Angriffen

Ein weitere Folge aus der Integration dieses neuen Dienstes sind Änderungen in der Ereignis-Detailansicht im Web-Frontend, in der nun alle von einem entfernten Host empfangenen Pakete inklusive Details (TCP-Flags, Nutzdaten) eingesehen werden können.

Sonstiges Weitere erwähnenswerte Änderungen sind die kontinuierliche Aktualisierung aller Build-Skripte, so wurde beispielsweise die Abhängigkeit von einer Python-Bibliothek, die inzwischen auch im offiziellen Debian-Repository verfügbar ist, entfernt. Die Skripte zum Generieren von BeagleBone-Images für SD-Karten wurden zudem so modifiziert, dass der Prozess auch in Container-Umgebungen wie *LXC*, das zur Entwicklung eingesetzt wird, laufen kann.

Weiterhin wurden einige graphische Elemente der Web-Benutzeroberfläche überarbeitet, darunter der nun global sichtbare und animierte Countdown bis zur nächsten automatischen Aktualisierung aller Anwendungsdaten. Ebenso wurde die Layoutstruktur der Webanwendung modernisiert, was sich beispielsweise darin äußert, dass nun beim Laden der Anwendung und all ihrer Abhängigkeiten im Browser entsprechendes Feedback zu sehen ist.

3 Ausblick

Auf Basis der genannten Änderungen ist beabsichtigt, nun mit dem Testbetrieb zu beginnen. Hierfür ist es zunächst noch erforderlich, Skripte zum automatisierten Generieren von Systemabbildern für die derzeitige Zielplattform, den BeagleBone Black, zu erstellen. Dabei werden die offiziellen Build-Skripte des BeagleBone-Projektes und die im Rahmen der zweiten Phase des Forschungsprojektes erzeugten Debian-Pakete als Grundlage dienen und den Schritt zum nutzbaren System stark vereinfachen. Anschließend sind, neben den noch offenen Punkten aus der Anforderungsanalyse, ausführliche Tests hinsichtlich Funktionalität und auch Sicherheit der Sensorarchitektur durchzuführen, um einen reibungslosen und möglichst gefahrlosen Betrieb des Systems zu gewährleisten. Es ist zudem damit zu rechnen, dass sich aus den Erfahrungen des Testbetriebes eine Vielzahl weiterer Anforderungen und Wünsche für die kommende Projektphase ergeben werden.

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
227	- Settings module extended with maintenance options (remove all events + reset whole system) - Menu point „Einstellungen“ renamed to „Syste“m - Minor layout changes to ensure correct rows/columns (bootstrap style), to that the correct spacing is provided everywhere - Beanstalk image conversion worker changed to also work within constrained container environments that don’t run udev. It creates and deletes the necessary devices with dmsetup - Gruntfile now sets executable bits on some of the python scripts
228	- Docker build process adjusted to current project structure - Even more scripts are now made executable during the project build process (required by docker deployment) - Sensor polling script changed to account for “invalid„ server SSL certificates due to an invalid local sensor date
229	Added event filters and filter conditions to the server-side model
230	Event UI converted into a module
231	- Event filtering finished (GUI + API) for source ip and event classification, including packet counters per filter - Fixed the sensor status list display and a few other Views, that relied on ’appendHtml’, which was renamed to ’attachHtml’ in Marionette - Fixed the automatic refresh of the sensor list - Various small markup and syntax fixes
232	All controller methods reworked to support the ’new’ API, that combines all selection criteria within a single get() method, making the old all() method superfluous
233	Properly return a 403 HTTP response in case an unauthorized access is made to an API ressource
234	Restructuring of the whole project directory. Server and sensor files are now split up.
235	- Remaining templates that weren’t yet split into modules were extracted from the main template into their own template files - The root layout can now be requested via HoneySens.request(’layout’) - Show the application logo during the application loading process

236	Reworked the application root layout: the login view and the application layout view (which houses the navbar and the sidebar) are now properly attached to the root layout view, we don't need fancy hacks on the body tag anymore. Login and logout is now also fully managed by the controller. This also allows a proper "loading,, view for when all the JavaScript data is still loading.
237	Basic debian packaging scripts for the sensor software
238	Model refresh countdown in the GUI is now globally visible and more simplistic
239	Sensor Makefile updated with build instructions for kippo and first dionaea prerequisites
240	Global menu update: - Menu definitions within modules now support a 'priority' attribute for the item order and a 'items' attribute to define a hierarchy for submenus - Remaining modules updated to use that menu system
241	Sensor debian package build process updated to include dionaea
242	Sensor software refactored, including major changes to the debian packaging script: includes init.d script, cron script, post-install hooks and a lot of dependencies. Changes to the existing projects dionaea and kippo are now just a set of patches applied during the build and install process. Removed some BeagleBoard-specific tweaks that have to be included in the platform-specific makefile later.
243	The debian sensor packaging scripts now install dionaea correctly, broken symlinks fixed
244	- kippo and dionaea modules rewritten to fetch their config values directly from the global HoneySens config file - kippo and dionaea modules disable themselves if no HoneySens config was found - rsyslog config file included (required for passive scan mode) - Fixed a bug that prevented successful package configuration due to a missing HoneySens config - Sensor configuration script updated to work on systemd architectures and to accept arbitrary file paths as arguments - Added error handling in case of a missing HoneySens config to the polling script and fixed a bug that prevented cron reconfiguration
245	Prototype of a new reconnaissance service added, that tries to collect the first data packet from each connection attempt and also in general logs more information than the older passive service. The service isn't yet integrated with the REST API.
251	Fixed a bug where all events couldn't be removed at once, because additional data was associated with them and caused constraint errors

256	Network Packets are now collected for each event by the recon service and displayed within the event details view in the web frontend. The recon service is now also multi-threaded for performance reasons. The pycrypto library is not part of the build process anymore, because it is available as a package within the Debian package repository.
257	“Flags,, column added for the packet details list (when viewing event details)
265	- Target port of individual packets is now recorded by the recon service and shown within the event detail dialog - Passive scan mode replaced by recon within the init script, syslog configuration that was a requirement of the old service removed - Filter conditions expanded to also include target port and protocol (TCP/UDP) - Filter condition dialog logic rewritten for better extensibility and redability
267	- The recon service now has a flooding protection and properly detects potential scan attempts - Cleaned up the modal event details view, packets are only shown when necessary
269	Fixed a bug where items weren’t added and removed properly to/from the event list, caused by the renaming of methods in a newer version of the marionette framework, making some overrides obsolete
270	Fixed a bug where the event details dialog wasn’t displayed correctly for scans
271	Fixed a bug where the dionaea service would report events about every single rejected packet that wasn’t handled by the recon service. The most likely cause, the pcap module, is now disabled. Also reduced the amount of logging on sensors.