Technische Universität Dresden

Fakultät Informatik

Professur für Datenschutz und Datensicherheit

Projekt HoneySens

5. Zwischenbericht

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

Stand: 18.07.2016

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.

Wesentliche Punkte der vorangegangenen Projektphase umfassten die Verbesserung und insbesondere Vereinfachung der Sensor-Server-Kommunikation, sowie eine umfassende Überarbeitung der Ereignisübersicht, um auch eine große Anzahl von Vorfällen dem Nutzer noch performant präsentieren zu können. Es wurde unterdessen deutlich, dass die vorgenommenen Änderungen bezüglich der Kommunikation über Proxy-Server hinweg ihren Zweck erfüllt haben und keine praktischen Probleme diesbezüglich mehr auftraten. In Gesprächen mit zahlreichen Administratoren wurde allerdings deutlich, dass noch Unsicherheiten und Vorbehalte bezüglich der Sicherheit des HoneySens-Systems selbst bestehen. Die Zuverlässigkeit dieser Plattform sicherzustellen genießt folglich oberste Priorität bei den weiterführenden Arbeiten und war auch Hauptgegenstand der fünften Projektphase. Ein ausführlicher Penetrationstest des HoneySens-Systems wurde hierfür als studentische Arbeit gestartet und parallel zu deren Unterstützung ausführliche Dokumentation über die API angefertigt. Als Resultat erfolgte eine Umstrukturierung und ausführliche Validierung aller Eingabeparameter der API.

Ein weiterer großer Punkt baut auf dem im letzten Bericht behandelten Abschnitt *Updateprozedur der Serversoftware* auf. Es hat sich in der Praxis herausgestellt, dass bei der Einrichtung eines neuen HoneySens-Servers diverse zu erledigende Punkte immer wieder neu erklärt und gesondert für die Nutzer dokumentiert werden müssen. Gleichzeitig wurde bei der Veröffentlichung einer neuen Software-Version deutlich, dass als Updateprozedur das bereits integrierte Schema-Update allein nicht ausreicht. Als Lösung wurde ein vollständiger Setup-Assistent integriert, der Nutzer bequem durch beide Prozesse führt und den Wechsel auf neue Software-Revisionen stark vereinfacht. Weitere Änderungen betreffen schließlich die Präsentation des Frontends, wovon insbesondere die neue Übersichtsseite („*Dashboard*“) dem Nutzer schnell ins Auge fällt.

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.

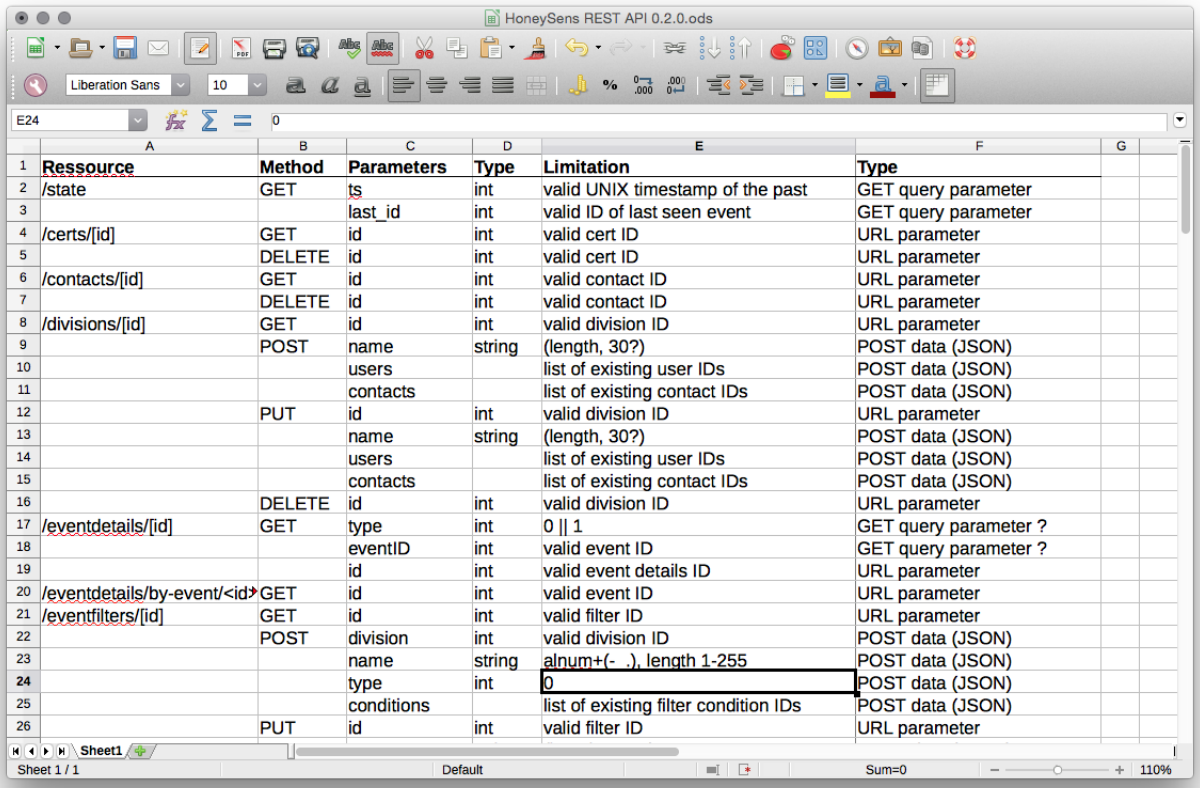
2 Sachstand

Dieser Abschnitt gibt einen Überblick über die vom 16. April 2016 bis zum 18. Juli 2016 verzeichneten Fortschritte am HoneySens-Projekt. Teilbereiche, die in den Anforderungen des ersten Projektberichtes genannt, aber hier nicht weiter beschrieben werden, sind Gegenstand der zukünftigen Weiterentwicklung.

Penetrationstest und Refaktorisierung der API-Schnittstelle Der Funktionsumfang der vom Server angebotenen und sowohl dem Nutzer als auch den Sensoren konsumierten API hat mit der Einführung neuer Funktionalitäten wie der filter- und sortierbaren Ereignisliste und der Mandantenfähigkeit erheblich zugenommen. Mit der steigenden Komplexität und der wachsenden Menge an Nutzereingaben, die vom System verarbeitet werden, wächst auch das Risiko von sicherheitsrelevanten Fehlern innerhalb dieser Schnittstelle. Eine umfassende Dokumentation und Verifikation aller übergebenen Parameter ist daher unumgänglich. Es wurde in diesem Zusammenhang zudem eine studentische Arbeit begonnen, die sich explizit mit einem *Grey-Box*-Penetrationstest des HoneySens-Systems beschäftigt. Bei derartigen Tests wird versucht, auf Basis der zugänglichen Dokumentation und anhand eines gesondert zu diesem Zweck zur Verfügung gestellten Laborsystems, Schwachstellen in der Implementierung aufzudecken. Ein Grey-Box-Test kombiniert verschiedene Eigenschaften von White- und Black-Box-Tests. Die interne Struktur der Anwendung (also der Quellcode selbst) steht dem Tester dabei nicht zur Verfügung, dafür aber ausführliche Dokumentation aller relevanten Schnittstellen und direkter Kontakt zum Entwickler für auftretende Fragen, sowie ein vollständiges Testsystem. Auf diesem Weg ist eine aufwändige Quellcodeanalyse nicht durchführbar, aber es können gezielt theoretische Schwachstellen der Implementierung herausgearbeitet und praktisch untersucht werden. Der Student erarbeitete hierfür anhand eines vom BSI veröffentlichen Standardmodells für Penetrationstest und der über HoneySens zur Verfügung gestellten Dokumentation einen genauen Testplan für die zu untersuchenden Komponenten.

In Vorbereitung des Penetrationstests wurde im Rahmen des Projektes eine umfassende Liste aller API-Endpunkte, deren Struktur, der erwarteten Parameter und deren akzeptierten Wertebereichen anhand des Quellcodes angefertigt (siehe Abbildung 1). Bei diesem Durcharbeiten des Codes wurden bereits eine Reihe von möglichen Problemen, wie beispielsweise eine nur unvollständige Überprüfung der akzeptierten Argumente, deutlich. Außerdem stellten sich einige der vorhandenen Funktionen als (nach einiger Refaktorisierung) inzwischen unnötig, bzw. auch destruktiv heraus, wenn im falschen Kontext angewendet. Positiv hervorzuheben ist aber, dass die auf diesem Wege gefundenen Probleme ausschließlich von bereits erfolgreich autorisierten Benutzern (mit den benötigten Zugangsberechtigungen) verursacht werden konnten. Weiterhin erforderte die anzufertigende Dokumentation eine Liste aller externen Abhängigkeiten und deren genutzte Revisionen, was sowohl alle eingebundenen Bibliotheken und Frameworks, sowie auch den serverseitigen Softwarestack (Web-Server, Datenbank-Server usw.) umfasst. Parallel dazu wurde zudem eine Liste aller für den HoneySens-Einsatz relevanten Softwarelizenzen erstellt, um rechtliche Klarheit für den Praxiseinsatz gewinnen zu können.

Der Penetrationstest selbst befindet sich noch im Gange und die exakten Ergebnisse der Arbeit stehen derzeit noch aus. Es wurde bis zum jetzigen Zeitpunkt kein schwerwiegender Fehler gefunden, der die Vertraulichkeit der im System gespeicherten Daten gefährden könnte. Trotzdem wurden bereits eine Reihe von Problemen offenbar. So ist es beispielsweise möglich, als Nutzer mit entsprechender Autorisierung gezielt einen DoS-Angriff auf ein System durchzuführen, indem bestimmte Parameter beim Hinzufügen eines weiteren Sensors unterschlagen werden.



HoneySens REST API 0.2.0.ods

Liberation Sans 10

E24 0

	A	B	C	D	E	F	G
1	**Ressource**	**Method**	**Parameters**	**Type**	**Limitation**	**Type**	
2	/state	GET	ts	int	valid UNIX timestamp of the past	GET query parameter	
3			last_id	int	valid ID of last seen event	GET query parameter	
4	/certs/[id]	GET	id	int	valid cert ID	URL parameter	
5		DELETE	id	int	valid cert ID	URL parameter	
6	/contacts/[id]	GET	id	int	valid contact ID	URL parameter	
7		DELETE	id	int	valid contact ID	URL parameter	
8	/divisions/[id]	GET	id	int	valid division ID	URL parameter	
9		POST	name	string	(length, 30?)	POST data (JSON)	
10			users		list of existing user IDs	POST data (JSON)	
11			contacts		list of existing contact IDs	POST data (JSON)	
12		PUT	id	int	valid division ID	URL parameter	
13			name	string	(length, 30?)	POST data (JSON)	
14			users		list of existing user IDs	POST data (JSON)	
15			contacts		list of existing contact IDs	POST data (JSON)	
16		DELETE	id	int	valid division ID	URL parameter	
17	/eventdetails/[id]	GET	type	int	0 || 1	GET query parameter ?	
18			eventID	int	valid event ID	GET query parameter ?	
19			id	int	valid event details ID	URL parameter	
20	/eventdetails/by-event/<id>	GET	id	int	valid event ID	URL parameter	
21	/eventfilters/[id]	GET	id	int	valid filter ID	URL parameter	
22		POST	division	int	valid division ID	POST data (JSON)	
23			name	string	alnum+(- _), length 1-255	POST data (JSON)	
24			type	int	0	POST data (JSON)	
25			conditions		list of existing filter condition IDs	POST data (JSON)	
26		PUT	id	int	valid filter ID	URL parameter	

Sheet1

Sheet 1 / 1 Default Sum=0 110%

Abbildung 1: API-Übersicht für den Penetrationstest

Aus den gewonnenen Erkenntnissen bei der Dokumentation der API, der Auflistung der externen Abhängigkeiten und den ersten Ergebnissen des Penetrationstests hat sich schließlich die Notwendigkeit ergeben, die API codeseitig klarer zu strukturieren und insbesondere eine separate Schicht zur Validierung aller Funktionsparameter in alle von API-Konsumenten aufrufbaren Methoden einzufügen. Hierfür wurde die *Respect Validation Engine*[1] als geeigneter Kandidat ausgewählt und in das Projekt integriert. Es ist mit diesem Framework leicht möglich, auch umfangreiche Regeln zur Überprüfung von Nutzereingaben durch die Verknüpfung von Einzelbedingungen zu notieren. Anschließend wurde die Liste aller API-Endpunkte schrittweise von Hand durchgearbeitet und alle Anforderungen (Existenz, Typen und Grenzwerte der Parameter) an die übermittelten Nutzerdaten in die entsprechenden Methoden integriert. In einem letzten Schritt wurde schließlich auch die Validierung im Frontend überarbeitet, um mit den teils neuen Schranken übereinzustimmen. Weitere Erkenntnisse aus dem Penetrationstest werden auch in Zukunft zeitnah in die Software integriert.

[1] `http://respect.github.io/Validation/`

Vereinheitlichung der API-Rückgabewerte Als der HoneySens-Forschungsprototyp entstand, folgten die Rückgabewerte der individuellen API-Methoden leider keinem einheitlichen Standard. Es wurde beispielsweise zunächst damit begonnen, auf jede Anfrage mit einem HTTP Status-Code 200 (*OK*) zu antworten und den eigentlichen Erfolg oder Misserfolg in einem JSON-Konstrukt mitzuteilen, das ebenfalls als Resultat zum Client gesendet wurde. Dieses Verfahren steht im Konflikt mit der üblichen Herangehensweise beim Entwurf von REST-APIs, erschwert aber insbesondere die Zusammenarbeit mit MVC[2]-Frameworks, die häufig im Frontend eingesetzt werden. So macht das für die HoneySens-Weboberfläche genutzte *Backbone*-Framework[3] den Erfolg von Operationen standardkonform am HTTP-Status-Code fest (2XX bedeutet hier Erfolg, 4XX oder 5XX Misserfolg). Um das Zusammenspiel dieser Komponenten zu erleichtern und einige der eingesetzten Workarounds entfernen zu können, wurden zusammen mit der im vorigen Abschnitt beschriebenen Refaktorisierung der API-Methoden die Rückgabewerte vereinheitlicht. So gibt die API nun ebenfalls mit Hilfe von HTTP Status-Codes Auskunft über den Verlauf der angefragten Operationen.

Dashboard Die System-Statusübersicht, die ein Nutzer nach dem erfolgreichen Anmelden am System präsentiert bekommt, hatte bisher nur *Proof-of-Concept*-Status und zeigte eher willkürlich auf einem Zeitstrahl den die über den Zeitraum des letzten Jahres in jedem Monat verzeichneten Ereignisse auf. Ebenfalls war eine kurze Auflistung der Sensoren mit den meisten aufgezeichneten Ereignissen und eine Liste der letzten fünf Vorfälle Gegenstand der Übersichtsseite. Es hat sich herausgestellt, dass diese starr präsentierten Daten in der Praxis eher selten ausgewertet wurden. Aus diesem Grund wurde ein neues Konzept für das sog. Dashboard erarbeitet und in die Praxis umgesetzt. Es beinhaltet ebenfalls wieder einen Zeitstrahl, auf dem Ereignisse pro Zeitabschnitt aufgetragen sind. Die präsentierten Daten können nun jedoch vom Benutzer beeinflusst werden: Mit einer Reihe von Auswahlfeldern ist es möglich, die dargestellten Ereignisse auf bestimmte Sensorgruppen, Monate oder Jahre einzugrenzen. An diese vom Nutzer festgelegten Parameter passen sich dynamisch auch alle weiteren dargestellten Widgets automatisch mit an. Derzeit umfasst dies eine Aufschlüsselung der Klassifikation aller im gewählten Zeitraum verzeichneten Ereignisse mit Hilfe eines Kreisdiagramms.

Setup-Prozess für Installation und Updates Im vorherigen Projektabschnitt wurde über die GUI ein generisches *Schema-Update* eingeführt, das Mechanismen des zugrundeliegenden Datenbank-ORM-Systems nutzt, um eine automatische Aktualisierung des Datenbankschemas herbeizuführen. Dies war, wie zu diesem Zeitpunkt bereits beschrieben, nur als Workaround und Vorarbeit für ein ausgereifteres Aktualisierungsverfahren zu verstehen. Da neue Versionen der Serversoftware tendentiell komplexere Modifikationen am Datenbestand als nur eine simple Aktualisierung des Schemas erfordern, wurde in der aktuellen Projektphase ein Setup-Assistent konzipiert und implementiert, der den Nutzer bei diesem Vorgang unterstützen soll. Zusätzlich wurde auch die Installation eines neuen Server-Systems berücksichtigt, bei dem der Administrator nun ebenfalls bei der Einrichtung der Weboberfläche Assistenz erhält.

Seitens der API existieren nun die zusätzlichen Endpunkte `/api/system/install` und `/api/system/update`, die vom Nutzer eine Reihe von für Installation und Update benötigte Daten entgegennehmen und anschließend den entsprechenden Prozess anstoßen. Im Falle der Installation umfasst das aktuell ein frei zu vergebendes Administratorpasswort (das an eine Mindestlänge von 6 Zeichen gebunden ist), den zu verwendenden Server-Endpunkt (typischerweise den *Common Name* des

[2] Model-View-Controller
[3] `http://backbonejs.org/`

Abbildung 2: Setup-Assistent im Frontend

verwendeten TLS-Zertifikats) und den Namen der ersten Sensorgruppe, die im Anschluss automatisch eingerichtet wird. Nach Angabe dieser Daten muss der Nutzer nach erfolgreichem Login lediglich eine Sensor-Firmware hinterlegen und kann dann direkt mit dem Hinzufügen erster Sensoren beginnen. Der Update-Prozess erwartet derzeit keine gesonderten Parameter, kann aber ausschließlich von autorisierten Administratoren angestoßen werden. Für alle anderen Benutzer ist ein Login nicht möglich, bis die Serversoftware den Aktualisierungsprozess abgeschlossen hat. Im Hintergrund erzeugt der Update-Endpunkt einen Beanstalk-Job, der von einem neuen Arbeiterprozess ausgeführt wird, der als Eingabeparameter die aktuelle Konfigurations-Revision und die Zielrevision erhält und dem alle relevanten Versionsunterschiede bekannt sind. Mit diesem Mechanismus ist auch ein inkrementelles Update über mehrere Versionen hinweg möglich. Wie bereits angedeutet, erfordert das Verfahren eine versionierte Server-Konfiguration, die im gleichen Schritt ebenfalls eingeführt wurde. Die genannten Änderungen ermöglichen es nun, auch umfassende Aktualisierungen, wie beispielsweise Veränderungen an der Struktur der Konfigurationsdatei oder Konvertierungen von Einträgen in der Datenbank inkrementell vorzunehmen.

Aus Nutzersicht ist im Falle einer Neuinstallation des Servers beim ersten Aufruf der Website oder im Falle einer Softwareaktualisierung nach erfolgreichem Login als Administrator ein Setup-Assistent sichtbar, der alle weiteren vorzunehmenden Schritte unmissverständlich beschreibt (vergleiche Abbildung 2).

Sonstiges Weitere erwähnenswerte Änderungen dieser Projektphase umfassen eine Reihe von visuellen Neuerungen, die die Arbeit mit der webbasierten Benutzeroberfläche angenehmer gestalten sollen. So wurde auf ein neues, modifiziertes Bootstrap-Thema umgestellt, das das Gesamterscheinungsbild der GUI mehr an die in Veröffentlichungen genutzten Farben, was insbesondere Rot- und Grautöne umfasst, anpasst. Dies erforderte eine Reihe von grafischen Änderungen innerhalb verschiedener Module des Nutzerfrontends. Auch das Projekt-Logo in der oberen linken Ecke der Navigationsleiste wurde durch eine neue SVG-Variante ersetzt, die auch auf hochauflösenden Geräten keine Pixelränder mehr erzeugt. Einhergehend mit diesen Änderungen wurde auch das Modul zur Verwaltung der Ereignisfilter aktualisiert und nutzt nun die *Backgrid*-Technologie (wie dies bereits bei der Ereignisliste selbst der Fall ist), was

Komfortfunktionen wie die spaltenweise Sortierung und leichter wartbare Filtermechanismen mit sich bringt. Die Formulare zum Hinzufügen und Bearbeiten von Filtern nutzen jetzt außerdem das flexible Overlay, das auch schon bei der Verwaltung von Sensoren eingesetzt wird und für mehr visuelle Übersichtlichkeit bei der Navigation durch das System sorgen soll.

Parallel zu den genannten Arbeiten wurden wieder eine Reihe von aufgetretenen Bugs behoben, darunter auch ein Fehler, der erst durch das Caching-Verhalten des *Internet Explorers* deutlich wurde. So wurde bisher Benutzern, die sich bereits erfolgreich angemeldet hatten (also ein gültiges Cookie besitzen) und die HoneySens-Webseite aufrufen, der für sie relevante Datenstand des Servers als JSON-Konstrukt direkt mit als Teil des HTML-Dokuments ausgeliefert. Dies stellte aber ein Sicherheitsrisiko da, wenn Tabs nach erfolgter Abmeldung offen gelassen wurden und nachfolgende Nutzer direkt aus dem Quellcode der Seite die Daten wieder auslesen konnten. Dieses konzeptionelle Problem wurde erst deutlich, als Nutzer im Internet Explorer nach erfolgter Anmeldung noch Restdaten aus einer vorherigen Session beobachten konnten. Dem Problem wurde begegnet, indem nun bei jedem Aufruf der Seite der aktuelle Datenstand explizit über AJAX-Requests vom Server neu angefragt werden muss und kein Ausliefern dieser Daten als Teil des HTML-Gerüsts mehr erfolgt.

3 Ausblick

Die nächsten Schritte umfassen den Abschluss und die Auswertung des Penetrationstests, sowie die Realisierung aller sich aus diesem ergebenden Änderungen. Besondere Priorität genießt neben der Ausweitung des Testbetriebes außerdem die Konzeption und Verwirklichung eines Signatursystems für Sensor-Firmware. Aktuell besitzt ein Angreifer mit Administorrechten die Möglichkeit, den Sensoren beliebige (bösartige) Software über das automatische Aktualisierungsverfahren unterzuschieben. Dieser Angriff erfordert zwar bewusst bösartig agierende Administratoren oder gestohlene Benutzerdaten, ist jedoch für den Einsatz innerhalb von kritischen Infrastrukturen trotzdem relevant. Es gilt daher, ein Verfahren zu entwickeln, das das Risiko eines solchen Vorfalls minimiert, indem alle betroffenen Parteien (insbesondere Firmware-Entwickler und Server-Betreiber) neue Firmware-Revisionen zunächst signieren müssen und Sensoren auch nur korrekt signierte Software akzeptieren.

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**
363	Event details view rearranged with content panes
364	- DoctrineExtensions added for extended DQL functionality - /stats endpoint added to API to retrieve general statistics for the dashboard - Chart.js updated to version 2.1.3 - underscore.js updated to version 1.8.3 - UI Dashboard modularized and rewritten from scratch, currently includes just an event timeline for 2016 - Fixed a bug that caused UI modules to start despite no user is logged in
365	- API stats ressource now supports filtering by user, year, month and division - UI dashboard now allows global filtering by division, month and year - Classification breakdown pie chart added to dashboard - Fixed a bug which caused the event list to be shown with old query parameters from previous invocations
367	- Bootstrap theme switched to „Simplex“ from bootswatch.com, followed by a lot of further modifications for visual clarity - CSS style format changed to be more compact - App layout modified to include a flexible sidebar to make more room for the actual content - JS library PieChartCountdown replaced by ProgressBar.js - Dashboard widgets now utilize global filters by division, month and year - PNG top-left-corner logo replaced by an SVG one - Spacing introduced between Event list filters - Fixed a bug that caused the dashboard to not load after a successful login under certain circumstances

369	- API restructured: all HTTP endpoint handlers moved into their respective controllers - Thorough validation added for all input parameters and matched against their respective frontend validators - Sanitized API return values: HTTP status codes are now properly used for successful or invalid calls - Documentation added to create and update methods with all relevant parameters - State resource added for accessing full system state or updates on that - RespectValidation framework integrated for API input validation - 'Install' and 'update' processes added. An associated 'setup' frontend module presents a wizard to users for these tasks. The update process itself is then handled by a separate beanstalk worker and responsible to update configuration and database state. - State data isn't directly written to HTML documents anymore and now only available via AJAX. This prevents data leaks and issues with Internet Explorer caching. - Bootstrap process simplified a lot - Server configuration is now versioned and supports the setting of the debug mode - Frontend modules are now named, which allows special routing conditions (like allowing access to the setup module for everyone) - Number of entries in event detail lists are now properly displayed - Refactored the frontend login process to allow easier hooking into it (e.g., for redirecting to the installer or updater) - Enabled german validation message feedback
370	Vector application logo added
371	- Compatibility with legacy configurations < 0.2.0 added to avoid update issues - Glyphicons added in woff2 format
372	- Event filter list module reworked using backgrid and the sliding overlay - Sidebar version indicator now properly displays the server software revision instead of a static value