HITSSSE
Höhere IT-Sicherheit durch
Sichere Software Entwicklung

IT-Sicherheit
IN DER WIRTSCHAFT

Demonstrator der Security Annotation

Veranschaulichung der Security Annotation an einem Beispiel

Gefördert durch:

Bundesministerium
für Wirtschaft
und Klimaschutz

aufgrund eines Beschlusses
des Deutschen Bundestages

1 Zusammenfassung

In diesem Artikel wird die Funktionsweise der Security Annotation anhand eines Demonstrators beschrieben. Dieser dient zum einen der Veranschaulichung des Konzeptes und zum anderen als Lern- und Testplattform. Deshalb befasst sich dieser Artikel mit dem Funktionsumfang des Demonstrators und geht auf ein konkretes Beispiel einer Security Annotation ein. Dabei wird das Beispiel zunächst analysiert und anschließend verbessert.

2 Das Konzept der Security Annotation

Das Konzept der *Security Annotation* ist eine spezielle, permanente Markierung im Quellcode, die das Risikomanagement erleichtern soll. Dazu wird der Quellcode mit zusätzlichen Informationen angereichert, sodass der Codebereich mit *Assets* und *potenziellen Schwachstellen* verknüpft werden kann[1].

Begriffsklärungen:

Definition:

Schützenswertes Gut (Asset): Bestände von Objekten (auch Daten), die einen bestimmten Zweck zur Erreichung von Geschäftszielen haben [4].

Schwachstelle (Vulnerability): Sicherheitsrelevanter Fehler eines IT-Systems. Dieser führt in Kombination mit dem Bedrohungspotential dazu, dass eine Bedrohung für ein System wirksam wird [4].

Potentielle Schwachstelle (Potential Vulnerability): Erweiterung des Begriffs der Schwachstelle für die Security Annotation. Dadurch lassen sich neben tatsächlichen Schwachstellen auch potentiell gefährdete Codeabschnitte markieren. Dies ist besonders hilfreich, wenn bei einer ersten Markierung eines Codeabschnitts die ausgehende Gefahr nicht vollständig geklärt ist.

Sicherheitssensitiver Bereich: Quellcode-Bereich innerhalb eines Softwareprojekts, welcher sicherheitsrelevante Elemente beinhaltet. Diese Bereiche können dabei Assets auf dem Quellcode repräsentieren oder die Bereiche markieren, in dem potentielle Schwachstellen auftreten können.

Security Annotation: Sprachunabhängige Quellcode Annotation, die es ermöglicht, sicherheitssensitive Bereiche dauerhaft zu markieren.

3 Ziel und Beschreibung des Demonstrators

Mithilfe eines praktischen Beispiels soll das Konzept der Security Annotation veranschaulicht werden. Um dies in einer abgekapselten und realistischen Umgebung zu gewährleisten, hat

[1] Für weitere Informationen über die Security Annotation, siehe mit Quellcode Annotationen Software sicherer machen

das HITSSSE-Projekt einen Demonstrator für die Security Annotation entwickelt. Das Ziel des Demonstrators ist es, Entwickler mit der Security Annotation vertraut zu machen. Außerdem sollen die Vorteile der Annotation an einem praktischen Beispiel veranschaulicht werden.

Um dies zu ermöglichen, liegt der Fokus des Projektes auf der Anwendung der Security Annotation anhand einer verbesserungswürdigen Codebasis. Die Codebasis ist dabei möglichst einfach gehalten und bietet zudem eine Nutzeroberfläche, um die Funktion des Demonstrators zu testen. Das Projekt basiert auf einer Java Anwendung in Form einer digitalen To-do-Liste. Diese Codebasis stammt von einem bestehenden Projekt [3] und wurde entsprechend angepasst.

Hierbei stellt das Programm eine Nutzeroberfläche bereit und verarbeitet Daten im Hintergrund. Sichtbar ist für den Anwender dabei lediglich das erstellen und verwalten von benutzerdefinierten To-dos. Die Demonstrator Architektur wird auf Abbildung 1 dargestellt und im nachfolgenden Abschnitt genauer beschrieben.

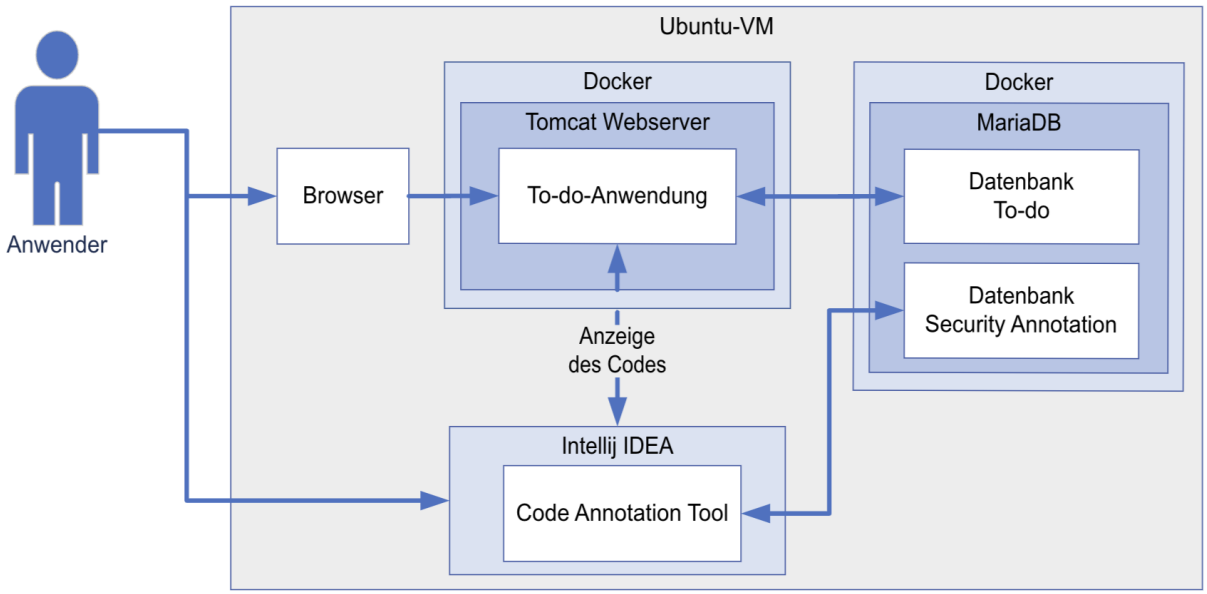

Abbildung 1: Darstellung der Demonstrator-Architektur

4 Aufbau des Demonstrators

Für das Testen des Demonstrators wird eine virtuelle Maschine (VM) bereitgestellt. Diese VM umfasst nachfolgende Elemente: [2]

- Betriebssystem (Ubuntu Version 20.04)
- Java Umgebung (Java Coretto-18.0.0.37.1)
- Git (Version 2.34.1)
- IntelliJ IDEA (Version 2021.3.3)
- HITSSSE Plugin (Version 213.7172.25)

[2] Die angegebenen Versionen beziehen sich auf den aktuellen Stand des Demonstrator-Projekts. Bei Änderungen wird das Dokument angepasst.

- Datenbank (MariaDB Version 10.6.7)
- Webserver (Tomcat Version 9.0.62)
- Docker (Version 20.10.23)
- Docker Compose (Version 2.15.1)

Zum Starten der VM wird ein Rechner mit x64 Architektur und VMware Workstation Player [12] benötigt. Nach der Installation wird das .ova Image (demonstrator.ova) mit einem Doppelklick in den VMware Workstation Player integriert und steht anschließend für den Einsatz bereit.

Nachfolgend werden die Anmeldedaten für die VM dargestellt:

Anmeldedaten:

Ubuntu:
Benutzername: *hitssse*, Passwort: *hitssse*

MariaDB (To-do-Datenbank):
Administrativer-Zugang: *root*, Passwort: *hitssse*
Gast-Zugang: *guest*, Passwort: *guest123*

Tomcat (Web-Anmeldung):
Benutzername: *admin*, Passwort: *admin*
Benutzername: *developer*, Passwort: *developer*
Benutzername: *guest*, Passwort: *guest*
Benutzername: *klaus*, Passwort: *klaus*

Nach dem Start der VM und dem erfolgreichen Anmelden, werden im Hintergrund alle notwendigen Docker-Container gestartet. Dadurch steht der Webserver und die dahinter liegende Datenbank zur Verfügung.

Für die Interaktion mit der Security Annotation wird zusätzlich IntelliJ IDEA benötigt, welches bereits vorinstalliert ist. Die Entwicklungsumgebung wird dann verwendet, um den Quellcode der To-do-Anwendung zu verwalten. Damit Optimierungen am Quellcode der Anwendung vorgenommen werden können, wurden absichtlich einige Schwachstellen eingebaut. Der Ausgangszustand der Anwendung wird im Branch *main* zur Verfügung gestellt. Um den Einstieg zu erleichtern, gibt es weitere Git-Branches und -Tags, die mögliche Security Annotationen und Codeverbesserungen (ohne Annotationen) bereitstellen. Die vorbereiteten Versionen sind in der Tabelle 4 dargestellt.

Git-Branch	Git-Tag	Beschreibung
main	unannotated_lineinput	Ausgangszustand des Projekts
-	unannotated_jdbc	Kommunikation mit JDCB ohne Security Annotation
unannotated	unannotated_jpa	Kommunikation mit JPA ohne Security Annotation
annotated	annotated_lineinput	Ausgangszustand des Projekts, jedoch inklusive gesetzter Beispiel-Annotationen

Im Ausgangszustand des Projektes erfolgt die Kommunikation mit der Datenbank über Lineinput. Diese Kommunikation kann als potentielle Schwachstelle angesehen werden. Zur Demonstration wurde diese Art der Kommunikation überarbeitet und kann unter den Git-Tags *unannotated_jdbc* und *unannotated_jpa* betrachtet werden. Damit soll gezeigt werden, wie ein

möglicher Umbau zur Vermeidung einer potentiellen Schwachstelle aussehen könnte. Mit dem Git-Branch *annotated_lineinput* werden beispielhafte Security Annotationen bereitgestellt. Diese können dann mit Informationen wie Assets oder potentielle Schwachstellen verknüpft werden.

Warum die Kommunikationsart eine potentielle Schwachstelle darstellt und welche Informationen den Security Annotationen hinzugefügt werden können, wird im nachfolgenden Abschnitt betrachtet.

5 Ein praktisches Beispiel

Im Folgenden wird der Anmeldevorgang beschrieben, um die Funktionsweise des Demonstrators zu veranschaulichen. Hierfür wird der Browser Firefox und das Entwicklungstool IntelliJ IDEA benötigt, die beide mit der Umgebung bereits installiert wurden.

Der Login-Prozess ist über den Firefox-Browser zugänglich. Das Anmeldefenster kann dabei durch folgende Adresse erreicht werden:

`http://localhost:8080/login/login.jsp`

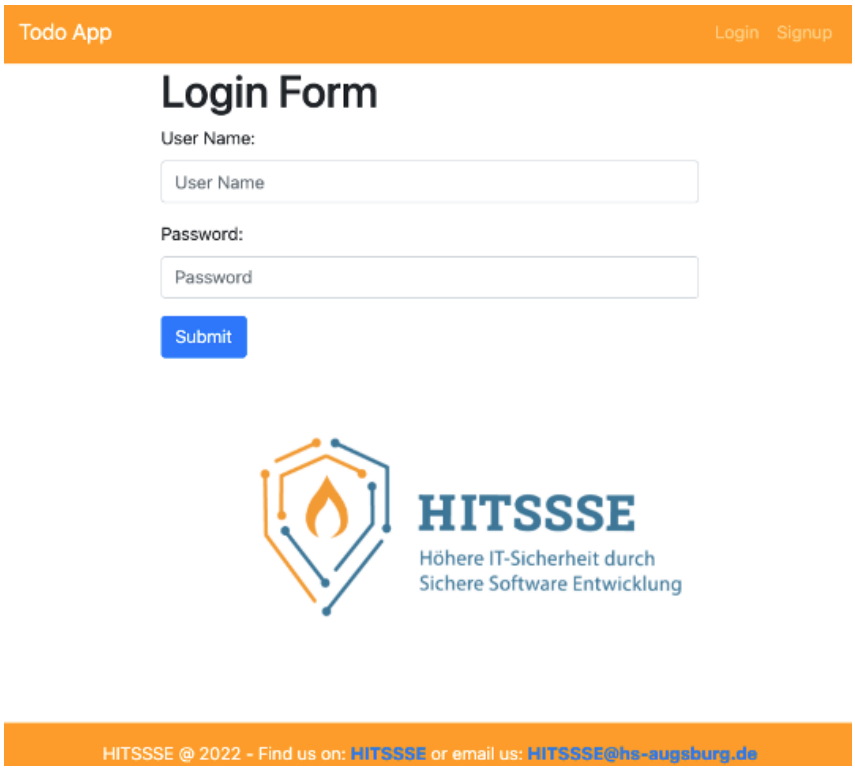

Abbildung 2: Anmeldefenster des Demonstrators

Auf der Abbildung 2 wird das Anmeldefenster gezeigt, welches Nutzername und Passwort für den Login-Vorgang benötigt. Nach Absenden der Login Daten überprüft die Anwendung die Anmeldedaten mit der Funktion „validate()“.

Der Codeabschnitt in Abbildung 3 implementiert diese Login-Funktion. Dabei werden die Benutzerdaten durch ein LoginBean angenommen und ungeprüft als SQL-Syntax konvertiert. Der SQL-Befehl wird dann direkt an die Datenbank gesendet und von dieser verarbeitet. Für den

```java
//!critFunction 17c5a9aa-5d5a-4c76-9c0d-f8975354db53
public boolean validate(LoginBean loginBean) throws ClassNotFoundException {
    boolean status = false;
    try {
        String cmd = "SELECT * FROM users WHERE username = ? and password = ?";
        cmd = cmd.replaceFirst( regex: "\\?", replacement: "\"" + loginBean.getUsername() + "\"");
        cmd = cmd.replaceFirst( regex: "\\?", replacement: "\"" + loginBean.getPassword() + "\"");
        List<String> res = DBConnection.getConnection(cmd);
        for (String line : Objects.requireNonNull(res)){
            if (line.trim().contains(loginBean.getUsername()) &&
                    line.trim().contains(loginBean.getPassword())) {
                logged_in_user = loginBean.getUsername();
                status = true;
                break;
            }
        }
    } catch (Exception exception){
        System.err.println(exception);
    }
    return status;
}
```

Abbildung 3: Login Funktion (Branch: *main*, angepasst)

Fall, dass die Login-Daten korrekt sind, enthält die Datenbankantwort den entsprechenden Eintrag. Dieser Eintrag wird anschließend in einer Schleife gefiltert. Im positiven Fall wird der Nutzer angemeldet und auf die nächste Webseite weitergeleitet. Falls die Daten jedoch inkorrekt sind, wird dem Nutzer eine Nachricht über die fehlerhaften Anmeldedaten angezeigt.

Diese Funktion repräsentiert einen kritischen Punkt bei der Nutzer-Authentifizierung und kann mit einer Security Annotation (siehe Abbildung 3) gekennzeichnet werden. Die Annotation ermöglicht das Hinzufügen von Meta-Informationen[3] wie z.B. dem *Asset* „Login Daten“. Die Überprüfung des Codes offenbart mögliche Sicherheitsrisiken, darunter SQL-Injection und fehlende Eingabevalidierung. Außerdem scheinen Passwörter in der Datenbank unverschlüsselt zu sein, da sie im Klartext an die Datenbank gesendet werden.

Zusammenfassung der gesammelten Informationen aus Abbildung 3:

Definition der Security Annotation:
Annotation der kritischen Funktion *validate()* durch *!critFunction*
Zuweisung einer eindeutigen Kennung: *17c5a9aa-3d5a-4c76-9c0d-f8975354db53*
Zuweisung von Assets[3]:
Login Daten
Zuweisung von potentielle Schwachstelle[3]:
SQL-Injection, Datenvalidierung und Passwort Hashing

Die Schwachstelle der SQL-Injection kann behoben werden, indem die Datenbank-Kommunikation verändert wird. Eine Möglichkeit, ist das Verwenden von Jakarta Persistence API (JPA)-Objekten,

[3]Weitere Informationen über die Funktionsweise der IDE und das Arbeiten mit der Security Annotation werden in dem Artikel „Beschreibung des IDE-Plugin“ beschrieben [6].

anstelle einer SQL-Syntax. Zusätzlich können alle Elemente der Datenbank bezogen werden, sodass die Nutzerdaten erst in der Schleife validiert werden. Auf diese Weise haben die eingegebenen Daten keinen Einfluss auf die Datenbank und die *potentielle Schwachstelle* der SQL-Injection kann damit behoben werden. Hierbei sollte jedoch beachtet werden, dass diese Optimierung Anpassungen an mehreren Stellen des Quellcodes erfordert und nicht nur an der auf Abbildung 3 dargestellten Codestelle. Die komplette Umstellung auf JPA ist in dem Demonstrator bereits vorbereitet und kann durch das Wechseln auf den Git-Branch „unannotated_jpa“eingesehen werden.

Im Rahmen des Umbaus wurden einige Teile der Anwendung überarbeitet. Hierbei wird ausschließlich auf den bereits vorgestellten Code-Abschnitt fokussiert, um die Übersichtlichkeit zu erhöhen.

```
//!critFunction 17c5a9aa-5d5a-4c76-9c0d-f8975354db53
public boolean validate(LoginBean loginBean) throws ClassNotFoundException {
    boolean status = false;
    try {
        User user = new User();
        List<User> userList = user.allEntries();
        for (User element : userList) {
            if (Objects.equals(element.getPassword(), loginBean.getPassword()) &&
                    Objects.equals(element.getUsername(), loginBean.getUsername())) {
                logged_in_user = element.getUsername();
                status = true;
                break;
            }
        }
    } catch (Exception exception){
        System.err.println(exception);
    }
    return status;
}
```

Abbildung 4: Login Funktion mit JPA (Branch: *unannotated_jpa*, angepasst)

Wie auf Abbildung 4 dargestellt, werden die Eingabedaten des Nutzers nun nicht mehr direkt an das Backend gesendet, sondern nur mit bereits vorhandenen Nutzerdaten verglichen. Durch diese Verbesserung wird zwar nicht jede evaluierte Sicherheitslücke der *Security Annotation* behoben, aber es konnte eine identifizierte Schwachstelle beseitigt werden.

Die Adressierung der *potentiellen Schwachstelle* ermöglicht nun die Kennzeichnung, dass diese *behoben* wurde. Diese Information wird in den Meta-Informationen der *Security Annotation* gespeichert. Im nächsten Schritt sollte die zuständige Person für die IT-Sicherheit die behebung des Fehlers überprüfen und die Schwachstelle als behoben *akzeptieren*. Dadurch ist die Schwachstelle in der *Security Annotation* behoben, bis sich die Code-Stelle ändert. Bei einer Änderung verliert die *potentielle Schwachstelle* den *akzeptiert* Status und wird standardmäßig wieder als *behoben* markiert.

6 Weiterführende Literatur

Die Funktionsweise der *Security Annotation* wird in detaillierter Form in dem Artikel *Mit Quellcode Annotationen Software sicherer machen* beschrieben [5]. Dabei wird vertieft auf das Konzept der Security Annotation, sowie das Zusammenspiel mit den Meta-Informationen eingegangen. Mit dem Artikel *Risiken dauerhaft mit Quellcode verbinden* wird das Zusammenspiel zwischen der Softwareentwicklung und dem Risikomanagement dargestellt [9]. Besonders hervorgehoben wird dabei die Bedeutung von Risiken für Unternehmen und die Vorteile einer inkrementellen Verwendung der *Security Annotation*. Dieser Artikel fokussierte sich auf die Sicht des Entwicklers, der die *Security Annotationen* vergibt und Meta-Informationen hinzufügt. Die Ansicht des Projekt-Managers wurde hierbei bewusst vernachlässigt. Für die Verwaltung der *Security Annotation* durch den Projekt-Manager hat das HITSSSE-Projekt ein *Asset Management Tool* entwickelt. Dieses Tool wird in dem Artikel *Grundlagen des Asset Management Tools* [7] beschrieben. Dabei wird dargestellt, warum es unterschiedliche Ansichten bei der *Security Annotation* gibt und ein Minimum Viable Product (MVP) beschrieben.

Im Demonstrator wurden bewusst Schwachstellen eingebaut, um künftige Optimierungsmöglichkeiten zu ermöglichen. Um sich vor Schwachstellen in der Software und unbeabsichtigten Fehlern bei der Entwicklung zu schützen, gibt es diverse Leitfäden. Mit dem Framework Software Assurance Maturity Model (SAMM) von OWASP, können Organisationen den Reifegrad ihrer Maßnahmen im Hinblick auf Anwendungssicherheit bewerten und überprüfen. SAMM bietet Organisationen dabei eine praktische Methode zur Überprüfung ihrer aktuellen Praxis, sowie zur Identifizierung von Bedrohungen und den daraus resultierenden, notwendigen Schutzmaßnahmen. Dieses Modell eignet sich besonders, da es modulbasiert aufgebaut ist und an spezifische Anforderungen und Bedürfnisse angepasst werden kann.
Auch das BSI stellt einen Leitfaden zur *Entwicklung sicherer Webanwendungen* [2] zur Verfügung. Dieser eignet sich besonders, da unterschiedliche Best Practices und Standards für die Softwareentwicklung betrachtet werden. Der Leitfaden befasst sich zu dem auch mit der Einführung von Sicherheit durch einen Software Development Lifecycle (SDL). Er enthält zudem Richtlinien für den Entwicklungs-/Implementierungsprozess, einschließlich Checklisten. Ein weiterer Leitfaden wurde im Rahmen des HITSSSE-Projekts entwickelt. Dieser *Secure Coding Guideline* [10] konzentriert sich auf die Programmiersprachen C/C++ im Embedded Systems Bereich. Dabei werden die beiden Standards MISRA und SEI CERT miteinander verglichen und deren Vor- und Nachteile untersucht.

Wie aus dem Artikel hervorgeht, ist die Identifizierung von Schwachstellen ein wichtiger Aspekt bei der Verbesserung der Sicherheit von Software und damit auch bei der Verwendung der *Security Annotation*. Hierfür gibt es Open-Source-Projekte wie *DEFECTDOJO* [1], das von OWASP angeboten wird und eine Sammlung von Schwachstellen-Scans zur Verfügung stellt. OWASP selbst bietet auch eine Liste aktueller Web-Schwachstellen an. Diese Liste wird regelmäßig aktualisiert und trägt den Namen OWASP-Top 10 [11]. Mit dem *Beispielkatalog für Assets und potentielle Schwachstellen* [8] stellt das HITSSSE-Projekt eine Sammlung von gängigen Meta-Informationen zur Verfügung. Ziel des Katalogs ist die Förderung des Verständnisses dieser Informationen und die Verringerung der Einstiegshürde in das Thema *Security Annotation*.

Impressum und Kontakt

Projekt HITSSSE – Höhere IT-Sicherheit durch Sichere Software Entwicklung

Immer mehr kleine und mittlere Unternehmen (KMUs) entwickeln Software für eigene Infrastrukturen oder Produkte. Hierbei herrscht meist ein hoher Zeitdruck und es stehen oft nur beschränkt personelle Ressourcen zur Verfügung. Oft werden inzwischen auch agile Softwareentwicklungsmethoden eingesetzt, die schnell einsetzbare Lösungen liefern sollen. Dadurch spielt die Sicherheit dieser Software oft eine untergeordnete Rolle, was sich letztendlich auch auf die IT-Sicherheit dieser Unternehmen und ihrer Kunden auswirkt. Im Fördervorhaben HITSSSE soll die IT-Sicherheit durch sichere Software Entwicklung für KMUs verbessert werden. Hierfür werden im Forschungsprojekt Handlungsempfehlungen sowie technische Hilfsmittel erstellt, die zuerst bei den assoziierten Partnern des Projekts konkret erprobt werden, um daraufhin generische Lösungsansätze für kleine und mittlere Unternehmen in Deutschland zu schaffen. Durch die Zusammenarbeit mit der Transferstelle „IT-Sicherheit in der Wirtschaft" soll die Breitenwirkung der entwickelten Angebote verstärkt werden. Gefördert wird das Projekt HITSSSE durch das Bundesministerium für Wirtschaft und Klimaschutz im Förderschwerpunkt Mittelstand-Digital. www.hitssse.de

Projektleitung

Prof. Dr.-Ing. Dominik Merli
Leiter HSA_innos
dominik.merli@hs-augsburg.de

Prof. Dr.-Ing. Alexandra Teynor
Leiterin HSA_ias
alexandra.teynor@hs-augsburg.de

Prof. Dr. Phillip Heidegger
HSA_ias
phillip.heidegger@hs-augsburg.de

Autoren

Raphael Mayr, M.Sc.
Wissenschaftlicher Mitarbeiter am HSA_ias
raphael.mayr@hs-augsburg.de

Daniel Haak, M.Sc.
Wissenschaftlicher Mitarbeiter am HSA_ias
daniel.haak@hs-augsburg.de

HSA_innos – Institut für innovative Sicherheit

HSA_innos hilft Unternehmen dabei, sich individuell zu schützen. Neben der Aus- und Weiterbildung von Sicherheitsexperten liegt der Schwerpunkt des Instituts auf der Entwicklung von Technologien und Prozessen für die IT-Sicherheit zur Anwendung in der Praxis. Zusammen mit HSA_innos schützen Unternehmen und andere Organisationen ihre Investitionen und Kunden vor digitalen Bedrohungen. Mehr Informationen zu HSA_innos finden Sie unter www.hsainnos.de.

HSA_innos
Institut für innovative Sicherheit

Hochschule Augsburg University of Applied Sciences

Institut für agile Softwareentwicklung

HSA_ias – Institut für agile Softwareentwicklung

Das Institut für agile Softwareentwicklung (HSA_ias) forscht in enger Zusammenarbeit mit Partnern aus Industrie und Wissenschaft zu den Schwerpunkten agile Softwareentwicklung, Programmiersprachen & Sicherheit, Prozessdigitalisierung sowie Anwendungen der KI. Die hierbei entstehenden Projekte decken eine breites Feld an Anwendungen ab, wie z.B. digitale Gesundheit, Produktionstechnik oder Digitalisierung der öffentlichen Verwaltung. Die Aus- und Weiterbildung von Software-IngenieurInnen für die Herausforderungen der Zukunft ist dabei ein zentrales Anliegen des Instituts.

Was ist Mittelstand Digital?

Das Mittelstand-Digital Netzwerk bietet mit den *Mittelstand-Digital Zentren*, der *Initiative IT-Sicherheit in der Wirtschaft* und *Digital Jetzt* umfassende Unterstützung bei der Digitalisierung. Kleine und mittlere Unternehmen profitieren von konkreten Praxisbeispielen und passgenauen, anbieterneutralen Angeboten zur Qualifikation und IT-Sicherheit. Das Bundesministerium für Wirtschaft und Energie ermöglicht die kostenfreie Nutzung und stellt finanzielle Zuschüsse bereit. Weitere Informationen finden Sie unter www.it-sicherheit-in-der-wirtschaft.de.

Mittelstand-Digital

Literatur

[1] Anderson, Greg and Weaver, Aaron and Tesauro, Matt and Scholten, Valentijn and Blaise, Fred. OpenSource Application Security Management, 2023. URL: https://www.defectdojo.com/.

[2] Bundesamt für Sicherheit in der Informationstechnik. Leitfaden zur Entwicklung sicherer Webanwendungen, 2013. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Webanwendungen/Webanw_Auftragnehmer.pdf?__blob=publicationFile&v=11.

[3] Fadatare, Ramesh. Todo Application, 2021. URL: https://github.com/RameshMF/todo-application-jsp-servlet-jdbc-mysql.

[4] Bundesamt für Sicherheit in der Informationstechnik. IT-Grundschutz Kompendium, February 2022. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Kompendium/IT_Grundschutz_Kompendium_Edition2022.pdf?__blob=publicationFile&v=5#download=1.

[5] Haak, Daniel and Mayr, Raphael. Annotieren von Code mit Hilfe der Security Annotation, 2023. URL: https://cloud.hs-augsburg.de/s/SxB8QSa4tZ6KDex.

[6] Haak, Daniel and Mayr, Raphael. IDE Plugin, 2023. URL: https://www.hitssse.de.

[7] Mayr, Raphael and Haak, Daniel. Asset Management Tool, 2023. URL: https://www.hitssse.de.

[8] Mayr, Raphael and Haak, Daniel. Beispielkatalog für Assets und PVs, 2023. URL: https://www.hitssse.de.

[9] Mayr, Raphael and Haak, Daniel. Risiken dauerhaft mit Quellcode verbinden, 2023. URL: https://cloud.hs-augsburg.de/s/dLkXet73wCnNL2f.

[10] Schloyer, Philipp. Secure Coding für Embedded Systems, 2022. URL: https://cloud.hs-augsburg.de/s/X8EDYHmkdBRXxGi.

[11] van der Stock, Andrew and Glas, Brian and Smithline, Neil and Gigler, Torsten. OWASP Top Ten, 2021. URL: https://owasp.org/www-project-top-ten/.

[12] VMware. VMware Workstation Player, 2022. URL: https://customerconnect.vmware.com/en/downloads/info/slug/desktopendusercomputing/vmwareworkstationplayer/160#product_downloads.