HITSSSE
Höhere IT-Sicherheit durch
Sichere Software Entwicklung

IT-Sicherheit
IN DER WIRTSCHAFT

Annotieren von Code mit Hilfe der Security Annotation

Strategien zur Identifizierung von sicherheitsrelevanten Codestellen

Gefördert durch:

Bundesministerium
für Wirtschaft
und Klimaschutz

aufgrund eines Beschlusses
des Deutschen Bundestages

1 Motivation

In dem Artikel „Security Annotation: Mit Quellcode Annotation Software sicherer machen“ [3] wurde die *Security Annotation* als Werkzeug für kleine und mittlere Unternehmen (KMU) vorgestellt. Sie soll insbesondere KMU dabei unterstützen, die IT-Sicherheit ihrer Produkte ressourcenschonend zu erhöhen. HITSSSE bietet ein Plugin, das mit allen JetBrain IDEs kompatibel ist, um mit der *Security Annotation* zu arbeiten. Mit diesem Plugin können Entwickler Annotationen effizient setzen und bearbeiten.
Dieser Artikel soll Entwicklern beim Umgang mit der *Security Annotation* helfen. Gleichzeitig soll vermieden werden, dass Entwickler die Annotation zu allgemein oder zu isoliert verwenden. Werden zu viele, eigentlich unkritische Codestellen annotiert, so verursachen diese Annotationen unnötigen Aufwand und verdecken die eigentlich wichtigen Stellen. Während unzusammenhängend verwendete Annotationen nur einzelne Codestellen abdecken, und so andere wichtige Codestellen aus dem Raster fallen.
Um den Entwicklern eine Hilfestellung zu geben, wird in diesem Artikel eine Strategie zum setzen der *Security Annotation* vorgestellt, an der sich Entwickler orientieren können. Diese Strategie wird dann in einem praktischen Beispiel veranschaulicht. Alle in diesem Artikel verwendeten Beispiele stammen aus dem von HITSSSE bereitgestellten Demonstrator [5].

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, so dass der Codebereich mit *Assets* und *potenziellen Schwachstellen* verknüpft wird. Weitere Informationen zu dem Konzept der *Security Annotation* sind in den folgenden zwei Artikeln zu finden [3, 6]

Definition:

Schützenswertes Gut (Asset): Bestände von Objekten (auch Daten), die einen bestimmten Zweck zur Erreichung von Geschäftszielen haben [1].
Schwachstelle (Vulnerability): Sicherheitsrelevanter Fehler eines IT-Systems. Dieser führt in Kombination mit dem Bedrohungspotential dazu, dass eine Bedrohung für ein System wirksam wird [1].
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.
Security Annotation: Annotation zum markieren von Quellcode-Bereichen innerhalb eines Softwareprojektes, welche sicherheitsrelevante Elemente beinhalten. Diese Bereiche können dabei *Assets* auf dem Quellcode widerspiegeln oder die Bereiche markieren, in dem *potentielle Schwachstellen* auftreten können.

3 Vorgehen beim Annotieren

Das Hauptmerkmal der *Security Annotation* besteht darin, dass *Assets* mit den zugehörigen Codestellen verknüpft werden können. Wie bereits in Kapitel 2 erläutert sind *Assets* bestände von Objekten (auch Daten), die einen bestimmten Zweck zur Erreichung von Geschäftszielen haben. Sie zeichnen sich vor allem dadurch aus, dass sie einen gewissen Wert für das Unternehmen haben und deshalb besonders schützenswert sind. *Assets* spiegeln sich im Quellcode durch klar abgrenzbare Pakete ab, die jeweils einzeln annotiert werden können.

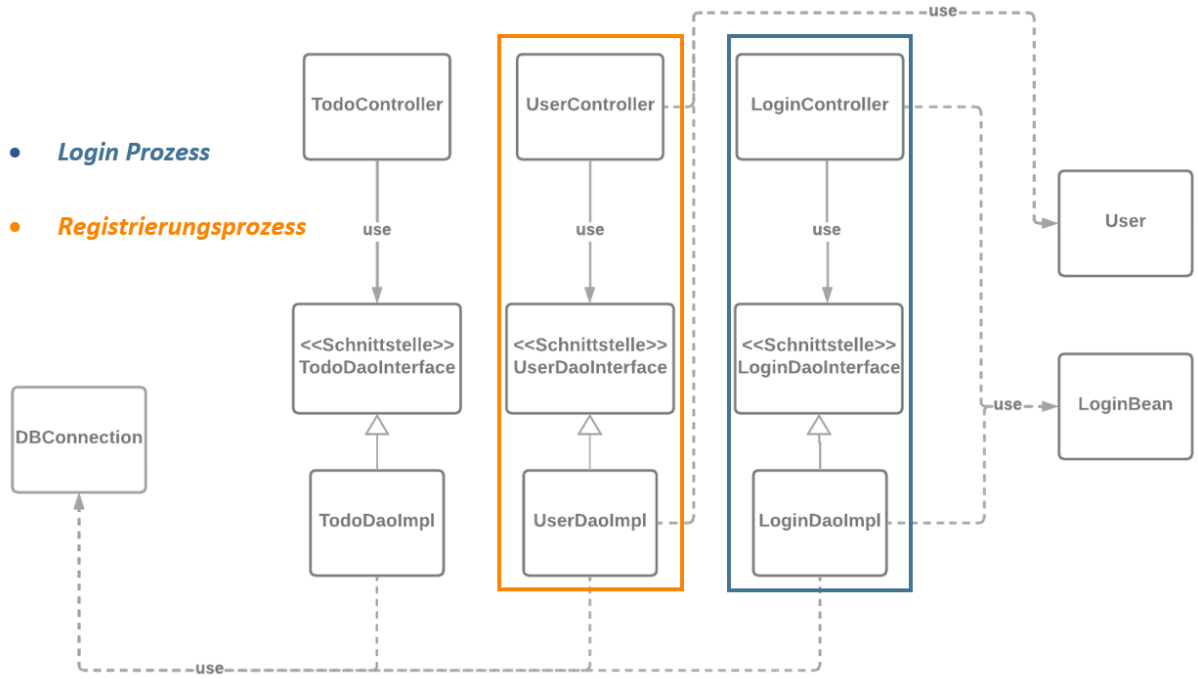

Abbildung 1: Klassendiagramm Demonstrator-Projekt

Ein Beispiel dafür sind die Login Daten der Kunden. In diesem Fall werden Codestellen für eine Annotation ausgewählt werden, die Login Daten der Kunden verarbeiten. In der Abbildung 1 wird der Aufbau der To-do-App aus dem Demonstratorprojekt [5] dargestellt. In der Abbildung sind der Registrierungs- und Loginprozess hervorgehoben, da in diesen beiden Prozessen die Login Daten der Kunden verarbeitet werden. Im *Asset*-basierten Ansatz wird üblicherweise prozessorientiert vorgegangen. Die vorgestellte Strategie sieht dementsprechend vor, diesen Prozessen zu folgen, die sich im Code durch Aufrufketten von Funktionen manifestieren.

Die Entwickler verfolgen diese Aufrufketten und annotieren alle Codestellen, die die Login Daten der Kunden verarbeiten. Dabei werden alle relevanten Funktionen berücksichtigt, beginnend mit der Eingabe im Frontend über die Weiterleitung ins Backend, die Verarbeitung und bis hin zur Ablage in der Datenbank. Insbesondere werden Funktionen von Interesse annotiert, welche die Daten nicht nur weiterleiten, sondern beispielsweise verschlüsseln, verifizieren, filtern oder anderweitig verarbeiten.

4 Praktisches Beispiel

Im vorangegangenen Abschnitt wurde ein Vorgehen zur Annotation von Quellcode vorgestellt. In diesem Kapitel wird dieses Vorgehen anhand eines Beispiels aus dem Demonstratorprojekt [5] veranschaulicht. Zunächst werden diese *Assets* identifiziert, entweder durch eine Risikoanalyse innerhalb des Unternehmens oder durch die Entnahme generischer Assets aus dem von HITSSSE bereitgestellten Katalog [4]. Anschließend werden die *Assets* nach Priorität geordnet.
Anschließend werden alle Codestellen identifiziert, die mit dem *Asset*, das die höchste Priorität hat, in Verbindung stehen. Als Beispiel wird hier das *Asset* Login Daten verwendet. Im Demonstratorprojekt werden die Login Daten im Registrierungs- und Loginprozess verarbeitet. Hauptsächlich sind dafür - wie in Abbildung 1 dargestellt - vier Klassen verantwortlich. Die Klassen UserController, UserDaoImpl, LoginController und LoginDaoImpl.
Die Login Daten werden auch in der Klasse User und der Klasse LoginBean zwischengespeichert. Diese beiden Klassen beinhalten nur Datenfelder und triviale Getter- sowie Setter-Funktionen. Würden diese Getter- und Setter-Funktionen signifikante Logik beinhalten, wären sie durchaus interessant als Kandidaten für die Annotation. Da sie in diesem Fall allerdings ausschließlich die Daten direkt weiterreichen sind sie im Rahmen dieses Beispiels uninteressant.
Die Klasse DBConnection steht mit beiden Prozessen in Verbindung. Da sie jedoch ausschließlich die Verbindung zur Datenbank aufbaut und nicht mit den Login Daten selbst in Berührung kommt, ist sie hier ebenfalls irrelevant. Sobald die relevanten Klassen identifiziert wurden, müssen die kritischen Codestellen innerhalb dieser Klassen annotiert werden. Dabei sind wie bereits erwähnt alle Codestellen von Bedeutung, welche die Daten aktiv verarbeiten.

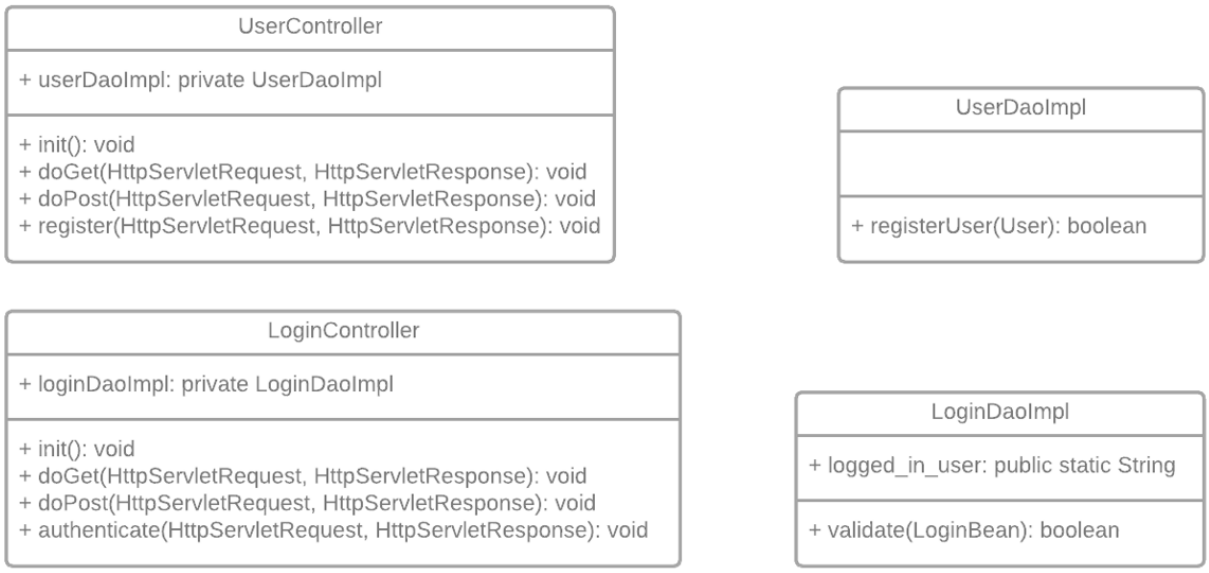

Abbildung 2: Auszug aus Klassendiagramm des Demonstrator-Projekts

4.1 UserController

Die Klasse UserController ist der Startpunkt des Registrierungsprozesses. In der Abbildung 2 ist eine Detailansicht der relevanten Klassen aus Abbildung 1 zu sehen. Die Klasse UserController empfängt die Http-Requests des Registrierungsformulars, extrahiert daraus die relevanten User-Daten und leitet sie an die Klasse UserDaoImpl weiter. Von den in Abbildung 2 dargestellten ist nur die register-Methode von Interesse. Die Anderen drei Methoden behandeln die Login Daten entweder nicht oder leiten diese direkt an die register-Methode weiter.

```
private void register(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
    try {
        boolean result = userDaoImpl.registerUser(new User(
                request.getParameter( s: "firstName"),
                request.getParameter( s: "lastName"),
                request.getParameter( s: "username"),
                request.getParameter( s: "password")
        ));
        if (result) {
            request.setAttribute( s: "NOTIFICATION",  o: "User Registered Successfully!");
        } else {
            request.setAttribute( s: "NOTIFICATION",  o: "Something went wrong!");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    RequestDispatcher dispatcher = request.getRequestDispatcher(REGISTER_PAGE);
    dispatcher.forward(request, response);
}
```

Abbildung 3: Quellcode der Methode register() im UserController (angepasst)

Abbildung 3 zeigt den Quellcode der register-Methode, welche die Nutzerdaten aus der Http-Anfrage extrahiert und an die UserDaoImpl-Klasse weiterleitet. Im Unterschied zu der doPost-Methode, werden hier die Nutzerdaten erstmals direkt verarbeitet und nicht als Bestandteil der Http-Anfrage weitergeleitet. Diese Methode ist ein wichtiger Knotenpunkt, der als Ausgangspunkt für künftige Erweiterungen dienen kann, obwohl sie derzeit eher trivial ist. Daher kann es sinnvoll sein, die Funktion zu annotieren, auch wenn sie derzeit wenig Fehlerpotenzial aufweist.

4.2 UserDaoImpl

Gemäß Abbildung 2 besitzt die Klasse UserDaoImpl lediglich eine Methode. Diese Funktion erhält die Benutzerdaten einschließlich der Anmeldedaten und speichert diese in der Datenbank. Die Implementierung ist in Abbildung 4 zu sehen. Nach erfolgreichem Speichern wird der Benutzer registriert. Aus zwei Gründen empfiehlt es sich, diese Methode zu annotieren. Zum einen greift sie im Allgemeinen auf die Datenbank zu und stellt somit eine kritische Schnittstelle dar, die fehleranfällig sein kann und ein lohnenswertes Angriffsziel darstellt. Zum anderen ist diese Funktion explizit anfällig für SQL-Injektionen und stellt somit eine konkrete Schwachstelle für das *Asset* dar, die behoben werden sollte.

```
public boolean registerUser(User user) throws ClassNotFoundException {
    boolean result = false;
    try {
        String cmd =  "INSERT INTO users (first_name, last_name, username, password) VALUES (?, ?, ?, ?);";
        cmd = cmd.replaceFirst( regex: "\\?",  replacement: "\"" + user.getFirstName() + "\"");
        cmd = cmd.replaceFirst( regex: "\\?",  replacement: "\"" + user.getLastName() + "\"");
        cmd = cmd.replaceFirst( regex: "\\?",  replacement: "\"" + user.getUsername() + "\"");
        cmd = cmd.replaceFirst( regex: "\\?",  replacement: "\"" + user.getPassword() + "\"");
        List<String> res = DBConnection.getConnection(cmd);
        result = res != null;
    } catch (Exception e) {
        System.err.println(e);
    }
    return result;
}
```

Abbildung 4: Quellcode der Methode registerUser() in der Klasse UserDaoImpl (angepasst)

4.3 LoginController

Die Klasse LoginController ist der Startpunkt des Loginprozesses. Sie funktioniert analog zu der UserController Klasse. Auch hier ist von den in Abbildung 2 dargestellten Methoden nur die authenticate-Methode von Interesse. Die Anderen drei Methoden behandeln die Login Daten entweder nicht oder leiten diese direkt an die authenticate-Methode weiter. In dieser werden die Login Daten aus den Http-Anfragen des Login Formulars extrahiert und an die Klasse LoginDaoImpl weitergeleitet, wo sie gegen die Einträge in der Datenbank verifiziert werden.

Die Abbildung 5 zeigt den Quellcode der authenticate-Methode. In ihr werden zunächst die Login Daten aus der Http-Request in eine LoginBean gespeichert. Anschließend wird die LoginBean an die validate-Methode aus der LoginDaoImpl-Klasse übergeben und das Ergebnis als Grundlage für die Weiterleitung des Nutzers genommen. Im Unterschied zu der doPost-Methode, werden hier die Nutzerdaten erstmals direkt verarbeitet und nicht als Bestandteil der Http-Anfrage weitergeleitet. Wie auch die register-Methode in der Klasse UserController ist diese Methode simpel gehalten, ist aber relevant, da sie eine Kommunikationsschnittstelle zwischen Front und Backend in den Prozess darstellt.

```
private void authenticate(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
    LoginBean loginBean = new LoginBean();
    loginBean.setUsername(request.getParameter( s: "username"));
    loginBean.setPassword(request.getParameter( s: "password"));
    try {
        if (loginDaoImpl.validate(loginBean)) {
            RequestDispatcher dispatcher = request.getRequestDispatcher(TODO_OVERVIEW);
            dispatcher.forward(request, response);
        } else {
            response.sendRedirect(LOGIN_CONTROLLER);
        }
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }

}
```

Abbildung 5: Quellcode der Methode authenticate() im LoginController (angepasst)

4.4 LoginDaoImpl

Die validate-Methode in der LoginDaoImpl-Klasse vergleicht Login Daten mit Einträgen in der Datenbank und behandelt den Nutzer als angemeldet, wenn ein passender Eintrag gefunden wird. Sie sollte aus zwei Gründen annotiert werden: Erstens, da sie auf die Datenbank zugreift und damit eine kritische Schnittstelle darstellt, die anfällig für Fehler sein kann und ein lohnendes Angriffsziel darstellt. Zweitens weist diese Funktion eine konkrete Schwachstelle in Form einer SQL-Injektion auf, die behoben werden sollte.

```
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)){
            // validate the login attempt of the identity
            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 6: Quellcode der Methode validate() in der Klasse LoginDaoImpl (angepasst)

5 Zusammenfassung

In diesem Artikel wurde eine Strategie vorgestellt, die Entwicklern als Leitfaden für die Annotation von Quellcode mit der *Security Annotation* dienen soll. Diese Strategie wurde anhand eines Beispiels veranschaulicht und kann je nach Projekt unterschiedlich angewendet werden. Entwickler müssen individuelle Entscheidungen darüber treffen, ob eine bestimmte Codestelle annotiert werden soll. Dabei kann die Strategie als Orientierungshilfe dienen, indem sie prüfen, ob eine Codestelle in Zusammenhang mit einem bestimmten *Asset* steht und Operationen darauf ausführt.

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

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] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grundschutz Kompendium*. 2022. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Kompendium/IT_Grundschutz/_Kompendium/_Edition2022.pdf?%5C_%5C_blob=publicationFile%5C&v=5#download=1 (besucht am 16. 12. 2021).

[2] Haak, Daniel and Mayr, Raphael. *Code Annotation Tool*. 2023. URL: https://www.hitssse.de/.

[3] Haak, Daniel and Mayr, Raphael. *Security Annotation*. 2023. URL: https://www.hitssse.de/.

[4] Mayr, Raphael and Haak, Daniel. *Asset/ potentielle Schwachstellen Katalog*. 2023. URL: https://www.hitssse.de/.

[5] Mayr, Raphael and Haak, Daniel. *Demonstrator der Security Annotation*. 2023. URL: https://www.hitssse.de/.

[6] Mayr, Raphael and Haak, Daniel. *Risiken dauerhaft mit Quellcode verbinden*. 2023. URL: https://www.hitssse.de/.