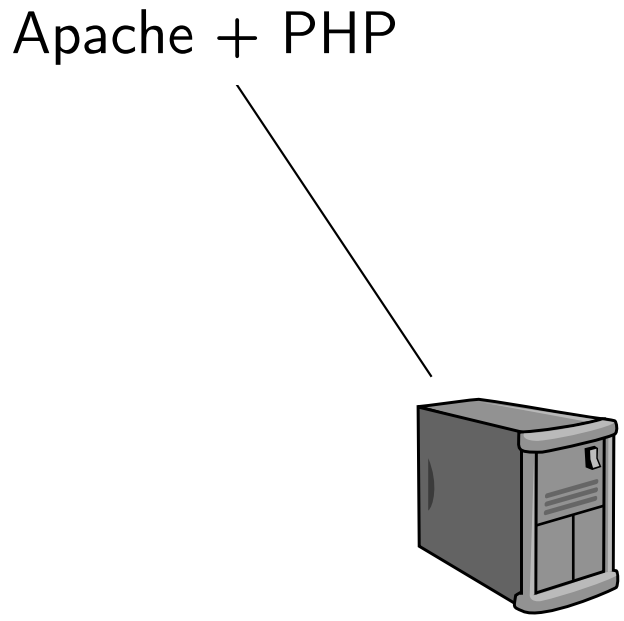
Informationssicherheit in Unternehmen – Qualitätskontrolle in der Praxis

geprüft, gestempelt, abgeheftet

Benjamin Kellermann

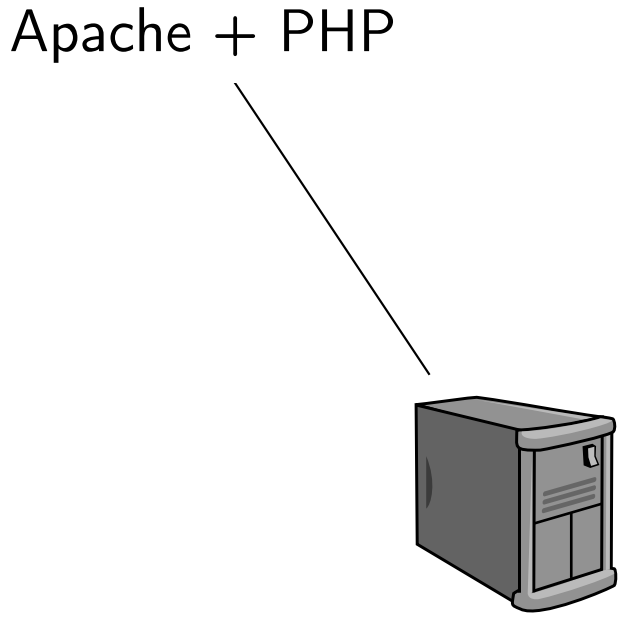
14. Oktober 2012

Motivation



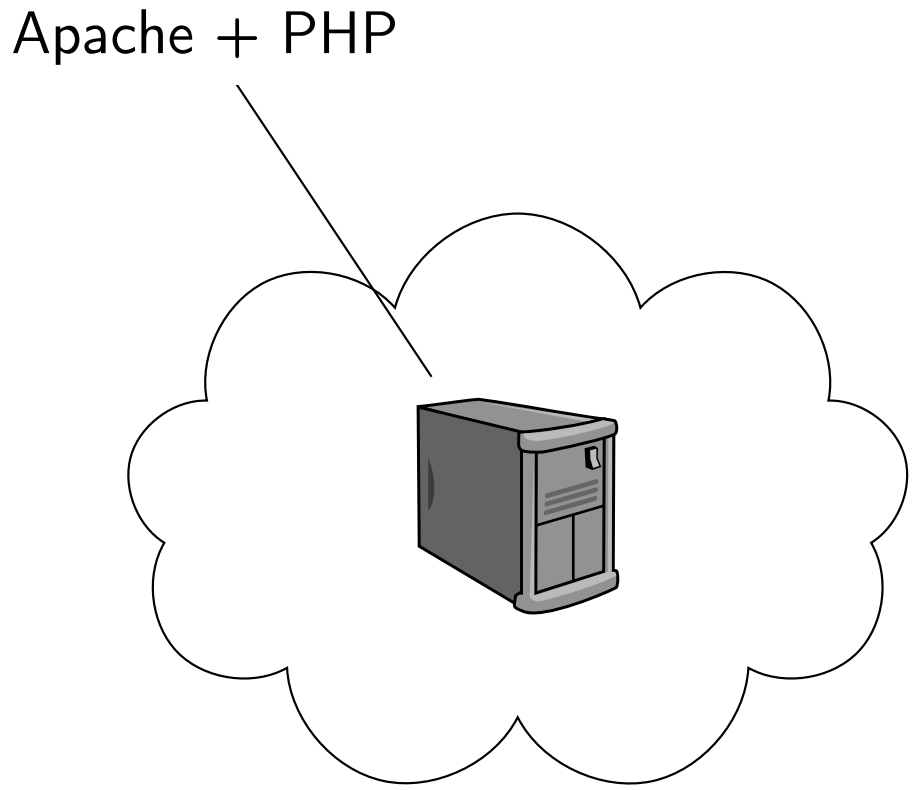
Motivation

- super Webapp



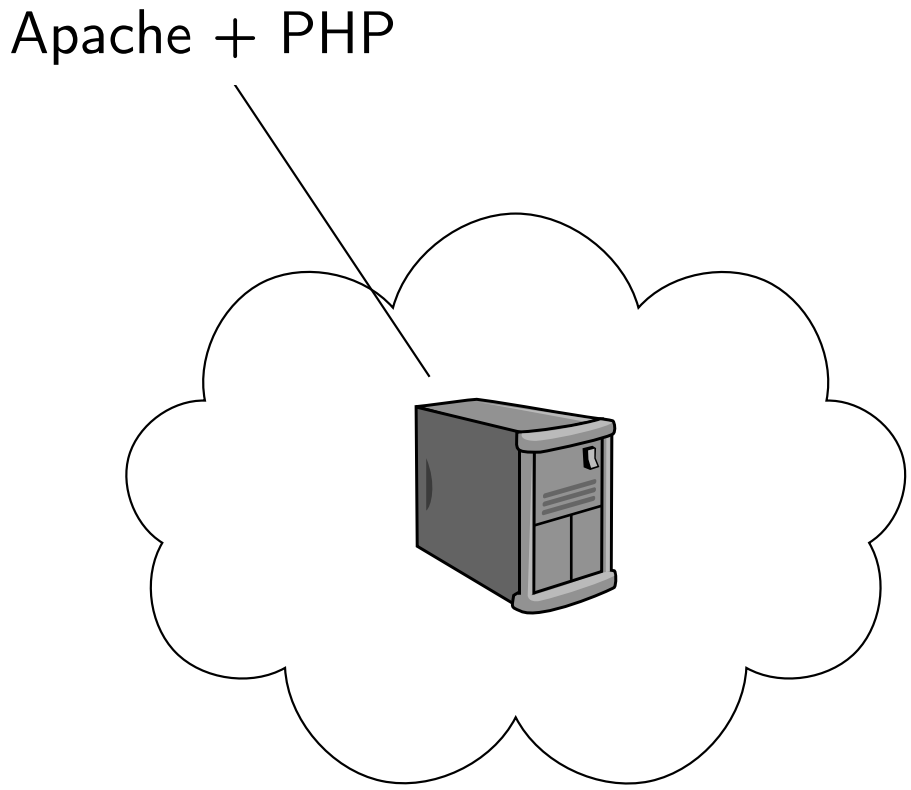
Motivation

- super Webapp



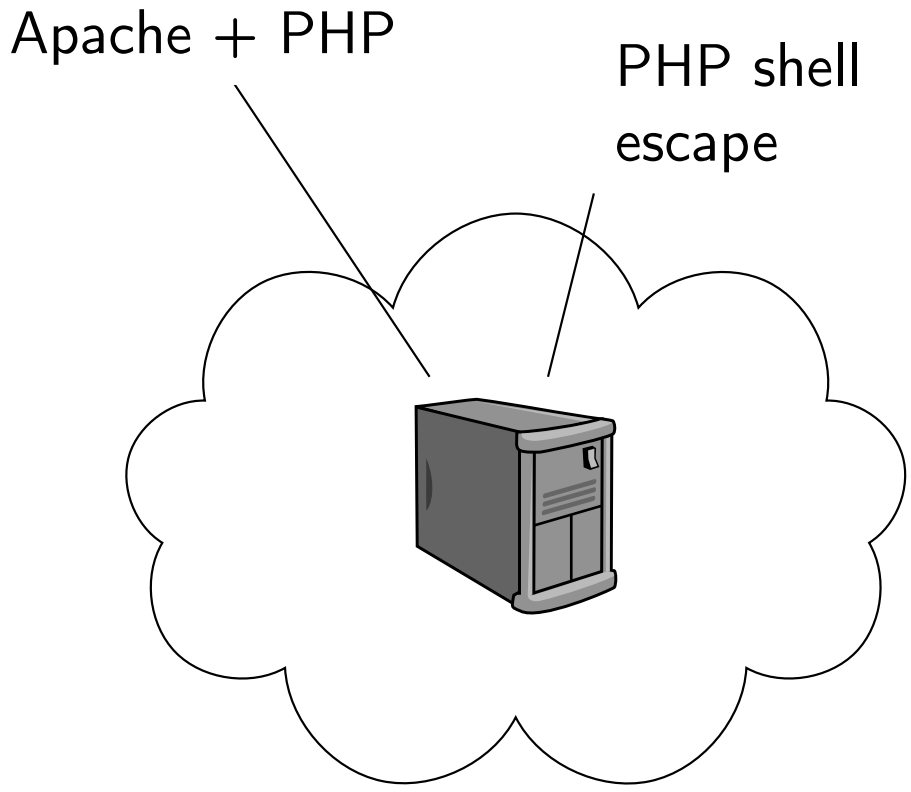
Motivation

- super Webapp
- Startup gegründet



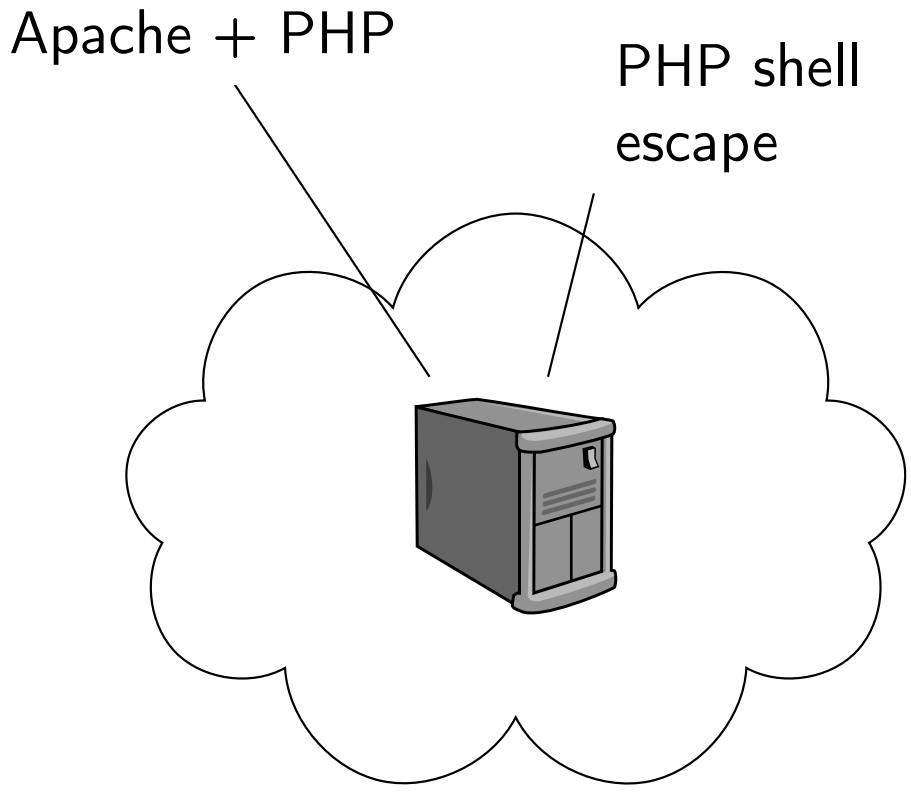
Motivation

- super Webapp
- Startup gegründet



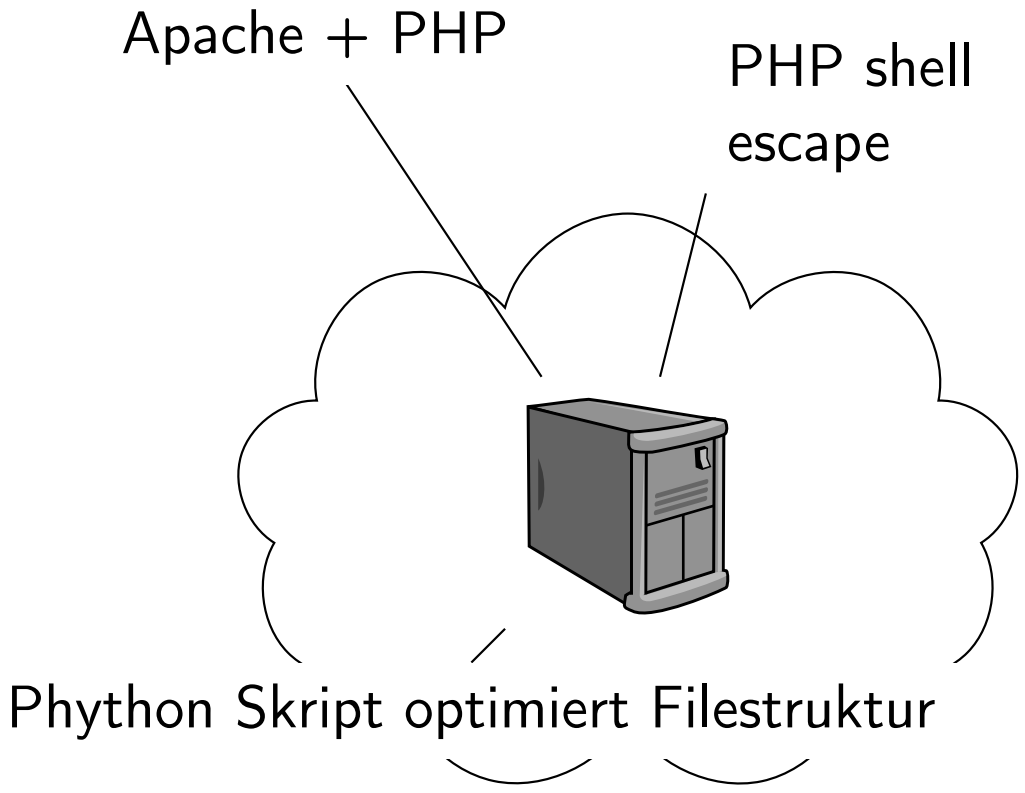
Motivation

- super Webapp
- Startup gegründet
- 100.000 Kunden-datensätze in 3 Monaten



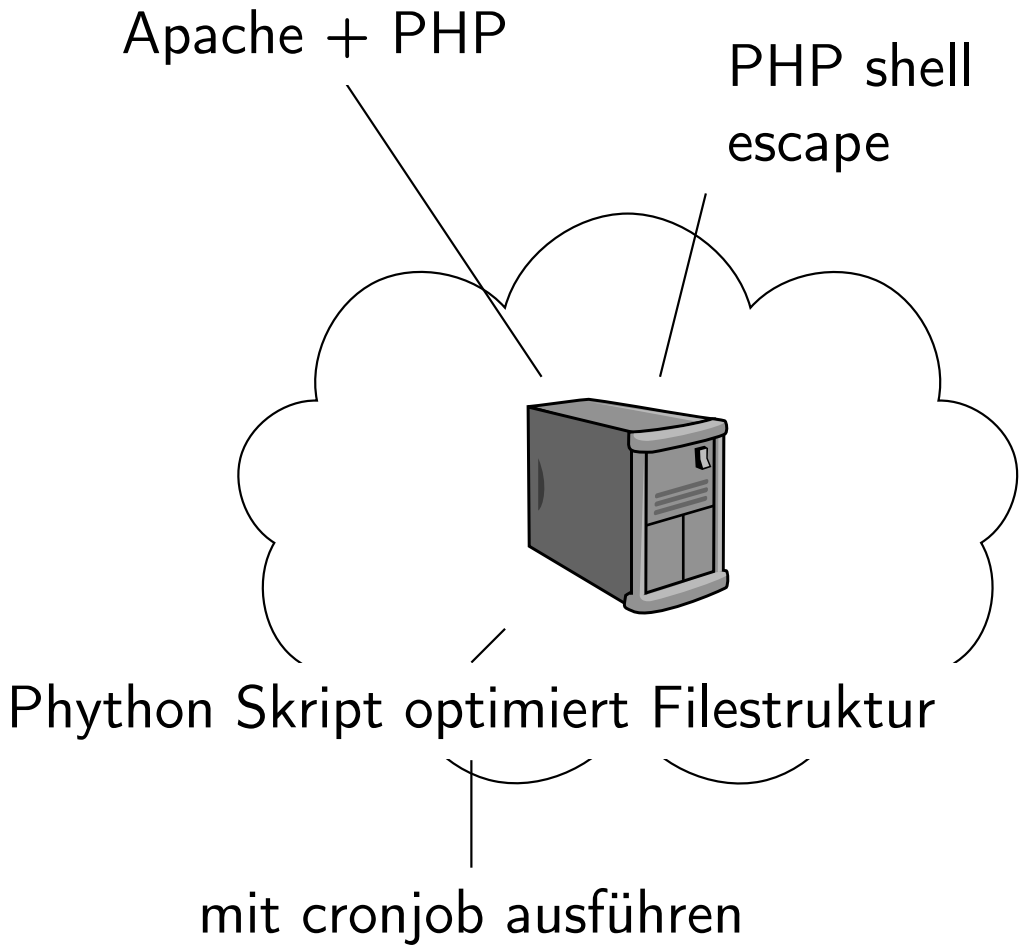
Motivation

- super Webapp
- Startup gegründet
- 100.000 Kunden-datensätze in 3 Monaten



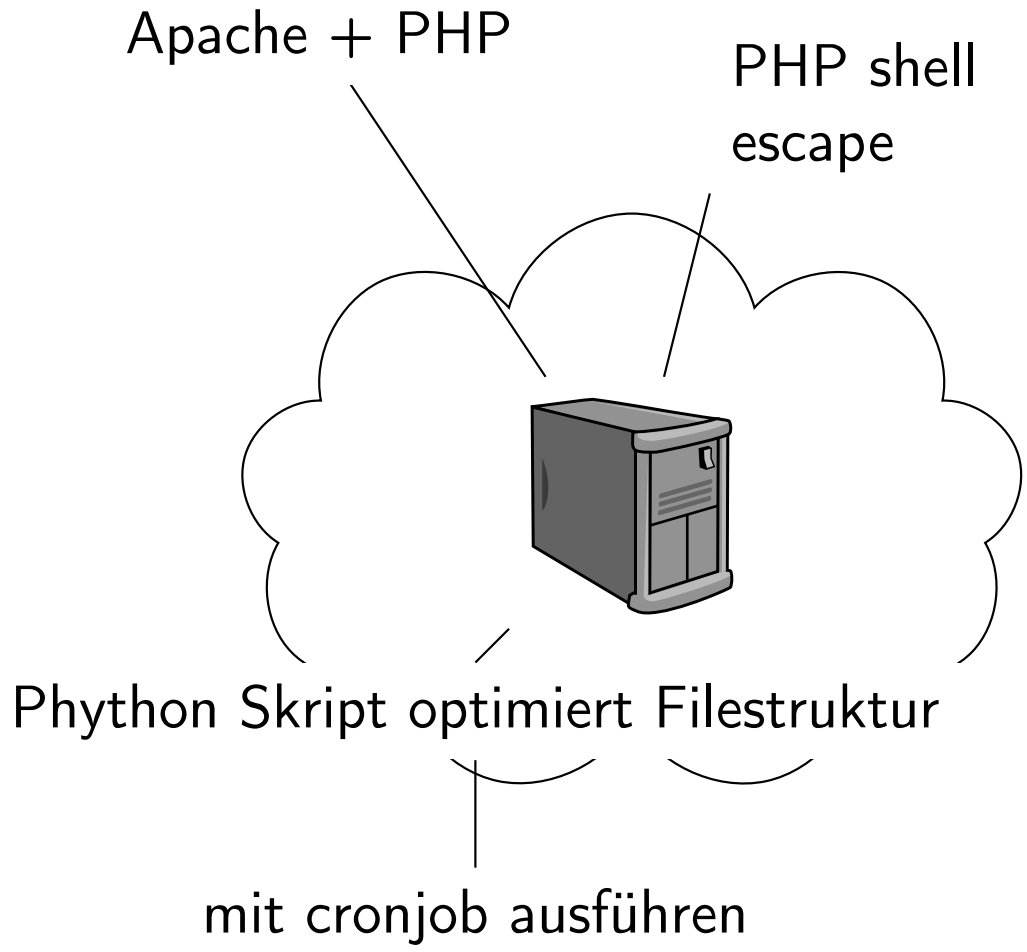
Motivation

- super Webapp
- Startup gegründet
- 100.000 Kunden-datensätze in 3 Monaten



Motivation

- super Webapp
- Startup gegründet
- 100.000 Kunden-datensätze in 3 Monaten
- Datenskandal bei der Konkurrenz



Standards

- ISO 27001
- BSI IT-Grundschutz
- Control Objectives for Information and Related Technology (CobiT)
- Information Technology Infrastructure Library (ITIL)

IT-Grundschutz

Beispiel: Webanwendungen (Entwurf vom Januar 2012)

Gefährdung

G 4.WA04 Unzureichende Validierung von Ein- und Ausgabedaten bei Web-Anwendungen

G 4.WA04 Unzureichende Validierung von Ein- und Ausgabedaten bei Web-Anwendungen

Unzureichende Validierung von Eingabedaten

Web-Anwendungen werden im Allgemeinen von generischen Clients (Web-Browsern) verwendet, sodass Benutzer beliebige Eingabedaten an den Server übermitteln können. Werden schadhafte Eingaben eines Angreifers von der Web-Anwendung verarbeitet, können möglicherweise Schutzmechanismen der Web-Anwendung umgangen werden.

Beispiele für Angriffe auf Web-Anwendungen, die auf einer unzureichenden Validierung von Eingabedaten beruhen, sind *SQL-Injection* (siehe G 5.131 *SQL-Injection*), *Path Traversal* und *Remote File Inclusion*. Diese Angriffe können Unbefugten Zugriff auf das Betriebssystem oder auf Hintergrundsysteme ermöglichen. Bei einem erfolgreichen Angriff können schützenswerte Daten unautorisiert ausgelesen oder manipuliert werden.

Unzureichende Validierung von Ausgabedaten

Nachdem die Web-Anwendung die Eingabedaten erfolgreich verarbeitet hat, werden üblicherweise wieder Daten ausgegeben. Die Ausgabedaten können entweder an den Browser des Benutzers (z. B. Statusmeldungen oder ein neuer Eintrag im Gästebuch) übermittelt oder an nachgelagerte Systeme weitergereicht werden. Werden die Daten vor der Ausgabe nicht ausreichend validiert, könnten die Ausgaben Schadcode enthalten, der auf den Zielsystemen entsprechend interpretiert oder ausgeführt werden.

Die folgenden Beispiele beschreiben mögliche Auswirkungen einer unzureichenden Validierung von Ein- und Ausgaben:

SQL-Injection

- Die Suchfunktion der Web-Anwendung verwendet die Eingaben der Benutzer ungefiltert zur Erzeugung von Datenbankabfragen. Dies kann ein Angreifer ausnutzen und eine Suchanfrage formulieren, die neben dem Suchbegriff zusätzliche Befehle für die Datenbank enthält. Durch das ungefilterte Einbetten der Suchanfrage in die Datenbankabfrage werden die Befehle von der Datenbank ausgeführt. Somit erhält der Angreifer direkten Zugriff auf die Datenbank.

Datei-Upload

- Eine Web-Anwendung bietet eine Funktion zum Datei-Upload an und schränkt diese auf gewisse Dateitypen ein. Zur Bestimmung des Dateityps überprüft die Web-Anwendung ausschließlich die Dateiendung und berücksichtigt dabei nicht den Inhalt der Datei. Wird eine erlaubte Dateiendung für den Upload verwendet, können so Dateien mit beliebigem Inhalt zum Server übermittelt werden.

Sanitizing

- Werden Eingabedaten durch die Filterkomponente automatisiert geändert und angepasst (Sanitizing), können die Daten durch gezielte Eingaben eines Angreifers von der Filterkomponente in einen Angriffsvektor überführt werden.

Unerwartetes Kodierungsschema

- Ein- und Ausgabedaten können in verschiedenen Kodierungen (z. B. UTF-8, ISO 8859-1) und Notationen (z. B. bei UTF-8 ist "." = "2E" = "C0 AE") vorliegen. Abhängig vom angewandten Kodierungsschema kann der gleiche Wert unterschiedlich interpretiert werden. Interpretiert die Filterkomponente die Daten anders als die verarbeitenden Komponenten der Web-Anwendung, so kann ein Angreifer schadhafte Daten (z. B. SQL-

Anweisungen) derart kodieren, dass sie bei der Filterung nicht erkannt werden. Somit werden die vom Angreifer schadhaften Daten an die verarbeitenden Komponenten weitergereicht und aufgrund der unterschiedlichen Interpretation ausgeführt.

Ungefilterter HTML-Code in Kommentar-Funktion

- Die Kommentar-Funktion einer Web-Anwendung erlaubt eine Formatierung der Texte durch HTML. Die Eingaben werden z. B. nicht auf spezielle HTML-Tags eingeschränkt, sodass ein Angreifer über diese Funktion beliebigen HTML-Code auf der Web-Anwendung platzieren kann. Dies kann ein Angreifer dazu nutzen, um Elemente der Webseite zu manipulieren oder zu überlagern und Benutzereingaben abzufangen (siehe G 5.WA18 *Clickjacking*).

IT-Grundschutz

Beispiel: Webanwendungen (Entwurf vom Januar 2012)

Gefährdung

G 4.WA04 Unzureichende Validierung von Ein- und Ausgabedaten bei Web-Anwendungen

Maßnahme

M 4.WA05 Umfassende Ein- und Ausgabevalidierung bei Web-Anwendungen

M 4.WA05 Umfassende Ein- und Ausgabevalidierung bei Web-Anwedungen

Verantwortlich für Initiierung: IT-Sicherheitsbeauftragter, Verantwortliche der einzelnen Anwendungen

Verantwortlich für Umsetzung: Administrator, Entwickler

Alle an die Web-Anwendung übergebenen Daten, unabhängig von Kodierung oder Form der Übermittlung, müssen als potenziell gefährlich behandelt und entsprechend gefiltert werden. Durch eine zuverlässige und gründliche Filterung der Ein- und Ausgabedaten mittels einer Validierungskomponente kann ein wirksamer Schutz vor gängigen Angriffen erreicht werden. Hierbei sollten sowohl die Eingabedaten von Benutzern an die Web-Anwendung als auch die Ausgabedaten von der Web-Anwendung an die Clients gefiltert werden. Dadurch wird sichergestellt, dass nur erwartete und keine schadhaften Daten von der Web-Anwendung verarbeitet oder ausgegeben werden.

Ist es für einzelne Funktionen notwendig, den Datenfilter weniger restriktiv zu setzen, muss dies explizit beim Zugriff auf die Daten definiert und dokumentiert werden. Zusätzlich ist es möglich, kontextsensitive Filter in der Geschäftslogik der Anwendung oder in den Hintergrundsystemen zu implementieren.

Für eine sichere Verarbeitung der Daten sollten folgende Punkte bei Umsetzung und Konfiguration der Validierungskomponente berücksichtigt werden.

Identifizierung der Daten

Damit die Ein- und Ausgabedaten einer Web-Anwendung umfassend validiert werden können, müssen zunächst alle zu verarbeitenden Datenstrukturen (z. B. E-Mail-Adresse) und die darin zulässigen Werte identifiziert werden. Für jede Datenstruktur sollte eine entsprechende Validierungsroutine umgesetzt werden. Neben der Datenstruktur sollte auch die Art und Weise der Datenverarbeitung erfasst werden (z. B. Weiterleitung an einen Interpreter, Rückgabe an den Client, Speicherung in einer Datenbank).

Berücksichtigung aller Daten und Datenformate

Die Validierungskomponente sollte alle zu verarbeitenden Datenformate und verwendeten Interpreter berücksichtigen. Typische Datenformate bei Web-Anwendungen sind z. B. personenbezogene Daten (Name, Telefonnummer, Postleitzahl), Bilder, PDF-Dateien und formatierte Texte. Typische Interpreter für Daten, die von Web-Anwendungen verarbeitet oder ausgegeben werden, sind z. B. HTML-Renderer, SQL-, XML-, LDAP-Interpreter und das Betriebssystem.

Techniken für die Validierung

Daten können durch unterschiedliche Techniken auf ihre Gültigkeit geprüft werden. So kann die Validierungskomponente den Wertebereich der Eingaben überprüfen oder es können beispielsweise reguläre Ausdrücke verwendet werden, um erlaubte Zeichen und die Länge der erwarteten Daten zu validieren. Die Gültigkeit von XML-Daten kann unter anderem mithilfe des entsprechenden XML-Schemas überprüft werden. Darüber hinaus stellen

Frameworks und Bibliotheken für gängige Datenformate entsprechende Validierungsfunktionen bereit.

Beispiele: Interpretierbare Zeichen

Die folgenden Zeichen werden gewöhnlich von in Web-Anwendungen eingesetzten Programmen interpretiert und können daher für das Einschleusen von schadhaftem Code genutzt werden. Aus diesem Grund sollten sie bei der Filterung berücksichtigt werden:

Nullwert, Zeilenvorschub, Wagenrücklauf, Hochkommata, Kommas, Schrägstriche, Leerzeichen, Tabulator-Zeichen, größer als und kleiner als, XML und HTML-Tags.

Diese Aufzählung erhebt keinen Anspruch auf Vollständigkeit. Zudem können die Interpreter-Zeichensätze (z. B. SQL-Syntax) bei unterschiedlichen Produkten variieren. Beispiele für kritische Zeichen werden im Abschnitt *Potentiell gefährliche Zeichen für Interpreter* in *Hilfsmittel zum Baustein Web-Anwendung* aufgeführt.

Validierung von Sekundärdaten

Neben den eigentlichen Nutzdaten (z. B. Formular-Parameter in GET- oder POST-Variablen) sind auch Daten anderer Herkunft (Sekundärdaten) zu validieren. Dazu zählen beispielsweise:

- Namen von Form-Variablen (Sie können ebenso wie der Wert der Form-Variablen beliebig manipuliert werden),
- HTTP-Header-Felder (Header-Felder in HTTP-Requests und -Responses sollten ausschließlich ASCII-Zeichen enthalten und z. B. keine Zeilenvorschubzeichen wie \r\n),
- SessionIDs (z. B. aus Cookies).

Automatisierte Aufrufe durch den Client z. B. durch Ajax- bzw. Flash-Skripte oder SOAP-Requests sind ebenfalls zu prüfen.

Daten aus Hintergrundsystemen

Bei den Hintergrundsystemen ist eine (gegebenenfalls erneute) Validierung der Daten vorzunehmen. Dies gilt auch dann, wenn Daten beispielsweise nach einem erfolgten Schreibvorgang in die Datenbank wieder ausgelesen werden, da die Daten auch in der Datenbank zwischenzeitlich geändert worden sein können.

Validierung unabhängig von der Quelle der Daten

Darüber hinaus sind Angriffstechniken bekannt, bei denen schadhafter Code über einen Kanal übermittelt wird, der nicht von der Web-Anwendung kontrolliert werden kann (z. B. FTP, NFS). Kann ein Angreifer über diese Dienste Dateien ändern oder erzeugen, die von der Web-Anwendung integriert werden, so kann Schadcode über diesen Umweg eingebettet werden. Bei dem sogenannten Cross-Channel-Scripting wird auf diese Weise Javascript-Code eingefügt, der ähnlich wie bei persistentem Cross-Site-Scripting vom Browser ausgeführt wird. Daher sollten unabhängig von der Quelle immer alle Daten der Web-Anwendung vor der Ausgabe an die Benutzer validiert werden.

Serverseitige Validierung

Üblicherweise greifen die Benutzer mit generischen Clients (z. B. Web-Browser) auf die Web-Anwendung zu. Diese Clients befinden sich nicht im Sicherheitskontext der Web-Anwendung, sondern stehen unter der Kontrolle der Benutzer. Die Datenvalidierung ist daher als serverseitiger Sicherheitsmechanismus auf einem vertrauenswürdigen IT-System umzusetzen.

Werden Daten zusätzlich durch Code von der Web-Anwendung clientseitig verarbeitet (z. B. Javascript-Code), so sollten diese Daten auch auf dem Client validiert werden. Die ausgelieferten Skripte der Web-Anwendung sollten hierbei die entsprechenden Validierungsroutinen mitliefern. Werden die Daten im nachgelagerten Verarbeitungsprozess an den Server gesendet, so ist zu beachten, dass die clientseitige Prüfung die serverseitige Validierung nicht ersetzen kann.

Validierungsansatz

White-List-Ansatz

Bei der Datenvalidierung wird zwischen dem White-List- und dem Black-List-Ansatz unterschieden.

Bei dem White-List-Ansatz werden ausschließlich solche Daten zugelassen, die in der Liste enthalten sind. Dabei werden, ausgehend von einer möglichst kleinen Zeichenmenge, Regeln erstellt, die Daten in einem festgelegten Zeichenraum zulassen und Daten zurückweisen, die abweichende Zeichen enthalten. Hierbei sollten komplexe Regeln durch die sequentielle Verwendung einfacher Regeln abgebildet werden.

Black-List-Ansatz

Dagegen werden bei einem Black-List-Ansatz solche Daten als unzulässig eingestuft und abgewiesen, die in der Liste enthalten sind. Alle Daten, die nicht explizit verboten sind, werden bei diesem Ansatz akzeptiert.

Bei dem Black-List-Ansatz besteht jedoch die Gefahr, dass nicht alle Variationen unzulässiger Daten berücksichtigt und somit erkannt werden. Daher sollte der White-List-Ansatz dem Black-List-Ansatz vorgezogen werden.

Kanonalisierung vor der Validierung

Daten können in verschiedenen Kodierungen (z. B. UTF-8, ISO 8859-1) und Notationen (z. B. bei UTF-8 ist "." = "2E" = "C0 AE") vorliegen. Abhängig vom angewendeten Kodierungsschema kann der gleiche Wert demnach unterschiedlich interpretiert werden. Findet eine Validierung der Daten ohne Berücksichtigung der Kodierung und Notation statt, so werden gegebenenfalls schadhafte Daten nicht erkannt und gefiltert. Daher sollten alle Daten vor der Validierung in eine einheitliche, normalisierte Form überführt werden. Dieser Vorgang wird als Kanonalisierung der Daten bezeichnet. Die so dargestellten Daten werden dann weiterverarbeitet.

Übermittllung des Kodierungsschemas

Darüber hinaus sollte das Kodierungsschema bei der Auslieferung von Daten durch die Web-Anwendung explizit gesetzt werden (z. B. über den Content-Type-Header: charset=ISO-8859-1).

Kontextsensitive Maskierung der Daten

Falls potentiell schadhafte Daten von der Web-Anwendung verarbeitet werden müssen (z. B. Zeichen mit einer Bedeutung für verwendete Interpreter) und somit eine Filterung nicht durchgeführt werden kann, müssen diese Daten maskiert und so in eine andere Darstellungsform überführt werden. In dieser maskierten Form werden die Daten nicht mehr als ausführbarer Code interpretiert. Da die Maskierung Interpreter-spezifisch ist, müssen alle verwendeten Interpreter berücksichtigt werden (z. B. SQL, LDAP). Die Maskierung muss demnach kontextsensitiv für das erwartete Ein- und Ausgabeformat und die Interpretersprache durchgeführt werden. Aufgrund der Komplexität und der spezifischen Anforderungen unterschiedlicher

Interpretersprachen wird empfohlen, für die Maskierung spezialisierte Bibliotheken einzusetzen.

Beispiele: Zu maskierende Zeichen

Es sollte eine Maskierung aller Zeichen vorgenommen werden, die als unsicher für den beabsichtigten Interpreter eingestuft werden. Dazu zählen z. B.

- unerwartetes JavaScript und HTML zur Auslieferung an den Client (Web-Browser),
- unerlaubt eingefügte SQL-Statements an die Datenbank (z. B. aus Eingaben in Formularfeldern),
- Befehle an das Betriebssystem (z. B. in manipulierten HTTP-Variablen).

Eine Maskierung kann durch eine Überführung der betroffenen Daten bzw. Metazeichen der jeweiligen Interpretersprache in sogenannte Zeichenreferenzen erfolgen. Das folgende Beispiel zeigt ausgewählte HTML-Zeichen mit den entsprechenden Zeichenreferenzen:

- & => &
- < => <
- > => >
- " => "
- ' => '

Hier ist darauf zu achten, dass &-Zeichen im ersten Durchlauf ersetzt werden, da dieses Zeichen in anderen Zeichenreferenzen als Metazeichen wiederverwendet wird.

Verwendung eines eigenen Markups zur Filterung von HTML-Tags

Filterung von HTML-Tags

Falls die Web-Anwendung HTML-Formatierungstags in Benutzereingaben erfordert (z. B. zur Formatierung von Benutzer-Beiträgen), sollten erlaubte HTML-Tags von problematischen Tags unterschieden und gefiltert werden (siehe auch Abschnitt *Kontextsensitive Maskierung der Daten*).

Dieser Ansatz ist mit dem hohen Risiko behaftet, problematische Tags (beispielsweise <script>) zu übersehen. Daher sollte der alternative Ansatz, für das Markup des Benutzers eigene Markup-Tags zu definieren (z. B. BBCode), vorgezogen werden. Diese Markup-Tags werden dann von der Anwendung in die zugehörigen HTML-Tags übersetzt. Herkömmliche Tags bzw. problematische Zeichen werden nach wie vor gefiltert.

Alternative Markup-Tags

Ein mögliches Verfahren, wenn ein einfaches Markup zugelassen werden soll, ist die Verwendung von { und } statt < und >. Fett würde dann als {F}**Dies ist fett**{/F} geschrieben und ein Bild könnte auf diese Weise platziert: {img src=/images/img.gif width=1 height=1 img}.

Hierbei darf die Umwandlung in HTML nicht einfach geschweifte Klammern durch spitze Klammern ersetzen, sondern muss jedes Tag als Ganzes ansehen:

- {img nach <img,
- img} nach >,
- src=Datei nach src="Datei" (wobei Datei zusätzlich zu filtern ist).

Wenn HTML-Tags zugelassen sind, ist grundsätzlich darauf zu achten, dass keine iFrame-Tags erlaubt sind. Mithilfe von iFrames können beliebige Inhalte in die Webseite eingefügt werden. Diese dürfen daher nicht genutzt werden können.

Behandlung von Fehleingaben (Sanitizing)

Anstatt Daten aufgrund eines unerwarteten Datenformats oder Zeichens abzulehnen, können Fehleingaben korrigiert und automatisch transformiert werden (engl. sanitize). Dadurch soll eine benutzerfreundliche Eingabe der Daten in unterschiedlichen Schreibweisen ermöglicht werden. Für eine Weiterverarbeitung lassen sich die Daten von unerwarteten Zeichen säubern (z. B. die Telefonnummer (0049)-201-01234567 kann in das nur aus Zahlen bestehende Format 004920101234567 überführt werden).

Eine Säuberung kann darin bestehen, Zeichen zu löschen, zu ersetzen oder zu maskieren (siehe auch Abschnitt *Kontextsensitive Maskierung der Daten*).

Sanitizing als potentielles Risiko

Beim Sanitizing besteht die Gefahr, dass Änderungen an den Daten zu einer neuen Komplexität, neuen Angriffsvektoren oder einer Missinterpretation führen. Daher sollte Sanitizing nach Möglichkeit vermieden und nur in Fällen angewendet werden, in denen ein Missbrauch des Sanitizing ausgeschlossen werden kann.

Falls die Web-Anwendung eine Korrektur der Daten erfordert, sollte die bewusste Manipulation von Daten (z. B. der SessionID durch einen Angreifer) nicht korrigiert, sondern abgelehnt werden. Darüber hinaus sollten Eingabedaten, die mit bestimmungsgemäßer Browser-Bedienung nicht eintreten können, grundsätzlich abgelehnt werden. Dazu zählen z. B.:

- Zusätzliche oder fehlende Formular-Parameter,
- Session-Cookies mit unerwarteten Zeichen oder ungültiger Länge,
- Unerwartete Werte bei der Übertragung von Formular-Parametern aus vordefinierten HIDDEN-, SELECT- oder CHECKBOX-Feldern,
- Der Übertragungsweg der Parameter (z. B. GET, POST, Cookie) stimmt nicht mit den Vorgaben der Anwendung überein.

Verschachtelte Angriffsvektoren

Bei einer Säuberung der Daten sollte die geschachtelte Eingabe von Angriffsvektoren berücksichtigt werden. Problematisch ist z. B. der auf den ersten Blick vernünftig erscheinende Filter s/<script>//g; (hier in Perl RegEx-Syntax geschrieben), um <script>-Tags im Eingabestrom zu löschen. Dieser kann jedoch mit einer geschachtelten Eingabe (z. B. <sc<script>ript>) umgangen werden. Es ist daher rekursiv zu filtern. Im Zweifelsfall sind die Eingabedaten abzulehnen.

Ablehnung der Daten im Fehlerfall

Grundsätzlich sollte bei einer Ablehnung der Daten die angeforderte Aktion ebenfalls abgebrochen und eine neutrale Fehlermeldung ausgegeben werden (siehe auch M 4.WA14 *Restriktive Herausgabe sicherheitsrelevanter Informationen bei Web-Anwendungen*). Bei Web-Anwendungen mit hohem Schutzbedarf sollte zusätzlich die Sitzung invalidiert werden.

Prüffragen:

- Werden alle Daten (Ein- und Ausgabedaten) und Datenströme der Web-Anwendung (z. B. zwischen Benutzer, Web-Anwendung und

Hintergrundsystemen) bei der Validierung berücksichtigt? {E: C/I}

- Werden auch Sekundärdaten (wie beispielsweise SessionIDs) bei der Validierung durch die Web-Anwendung berücksichtigt? {E: C/I}
- Führt die Web-Anwendung eine serverseitige Validierung der Daten auf einem vertrauenswürdigen IT-System durch? {E: C/I}
- Führt die Web-Anwendung vor der Validierung eine Kanonalisierung der Daten durch? {E: C/I}
- Findet in der Web-Anwendung eine kontextsensitive Validierung der Daten unter Berücksichtigung des erwarteten Interpreters der Daten statt? {E: C/I}
- Bei Web-Anwendung mit automatischer Behandlung von Fehleingaben (engl. *Sanitizing*): Wird die Behandlung von Fehleingaben sicher umgesetzt? {V. C/I}

Praxisumsetzung

hier: Telekom Stand August 2012

Eingabevalidierung

- Sämtliche Daten, die von einem Clientübertragen wurden, MÜSSEN von einer Web-Anwendung gemäß ihrer Spezifikation serverseitig validiert werden.

Praxisumsetzung

hier: Telekom Stand August 2012

Eingabevalidierung

- Sämtliche Daten, die von einem Clientübertragen wurden, MÜSSEN von einer Web-Anwendung gemäß ihrer Spezifikation serverseitig validiert werden.
- Die Eingabevalidierung einer Web-Anwendung SOLLTE über Whitelisting erfolgen.

Praxisumsetzung

hier: Telekom Stand August 2012

Eingabevalidierung

- Sämtliche Daten, die von einem Clientübertragen wurden, MÜSSEN von einer Web-Anwendung gemäß ihrer Spezifikation serverseitig validiert werden.
- Die Eingabevalidierung einer Web-Anwendung SOLLTE über Whitelisting erfolgen.
- Wenn in einer Web-Anwendung durch die Nutzer auf Dateien zugegriffen wird, DARF der Dateiname NICHT auf Benutzereingaben basieren.

Praxisumsetzung

hier: Telekom Stand August 2012

Eingabevalidierung

- Sämtliche Daten, die von einem Clientübertragen wurden, MÜSSEN von einer Web-Anwendung gemäß ihrer Spezifikation serverseitig validiert werden.
- Die Eingabevalidierung einer Web-Anwendung SOLLTE über Whitelisting erfolgen.
- Wenn in einer Web-Anwendung durch die Nutzer auf Dateien zugegriffen wird, DARF der Dateiname NICHT auf Benutzereingaben basieren.
- . . . und 20 weitere Anforderungen.

Risikoanalyse

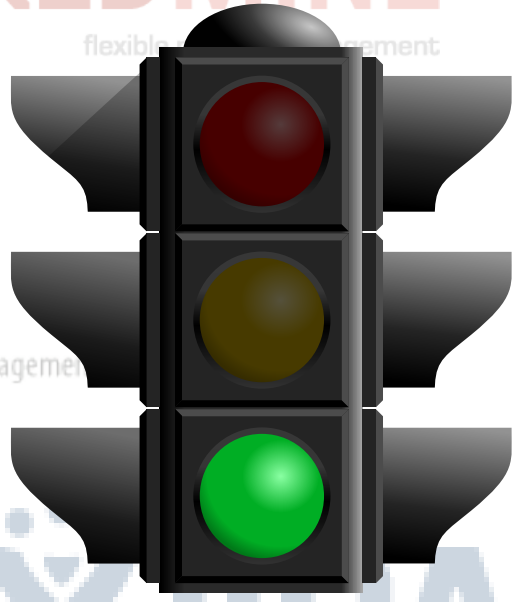
- Aufnahme von Requirementerfüllungen/-verletzungen

Tools

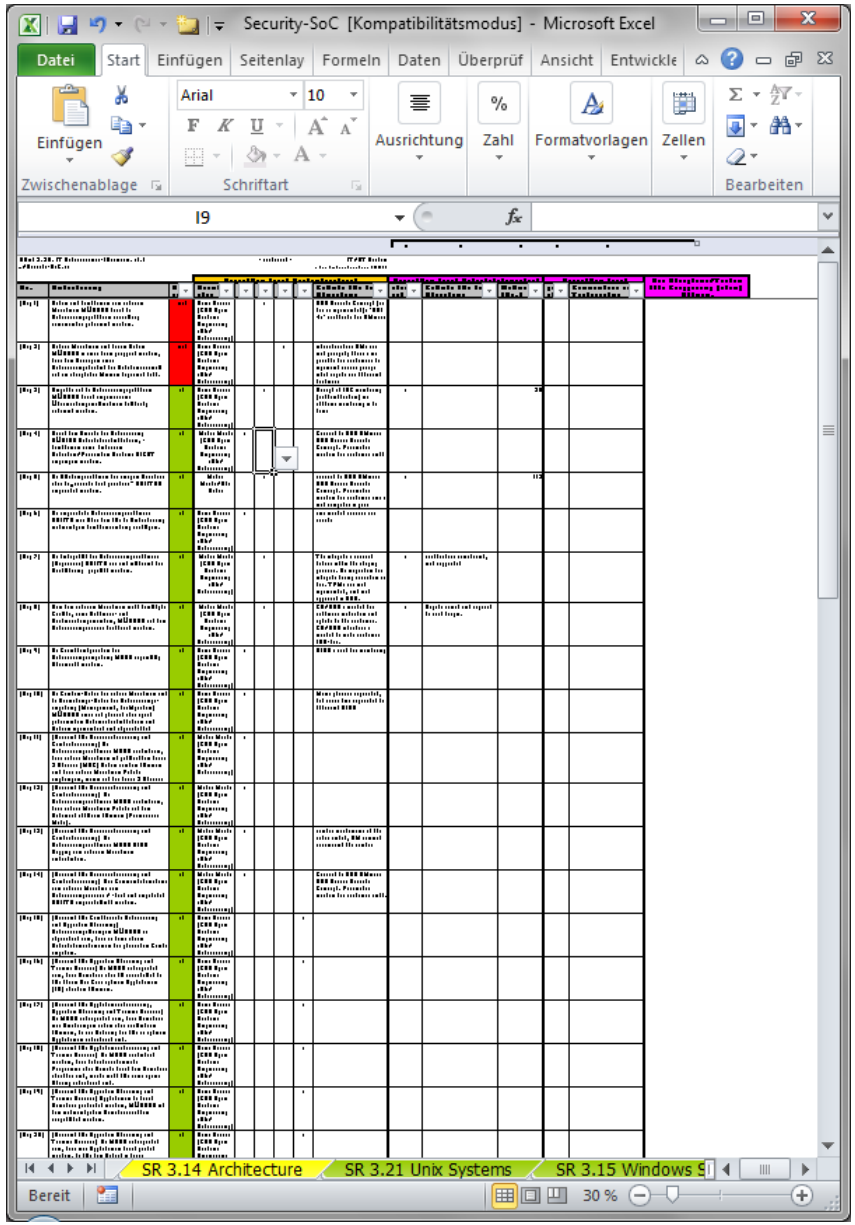

REDMINE
flexible project management
mantis
BUG TRACKER
trac
Integrated SCM & Project Management
Bugzilla
JIRA

Tools

Tools



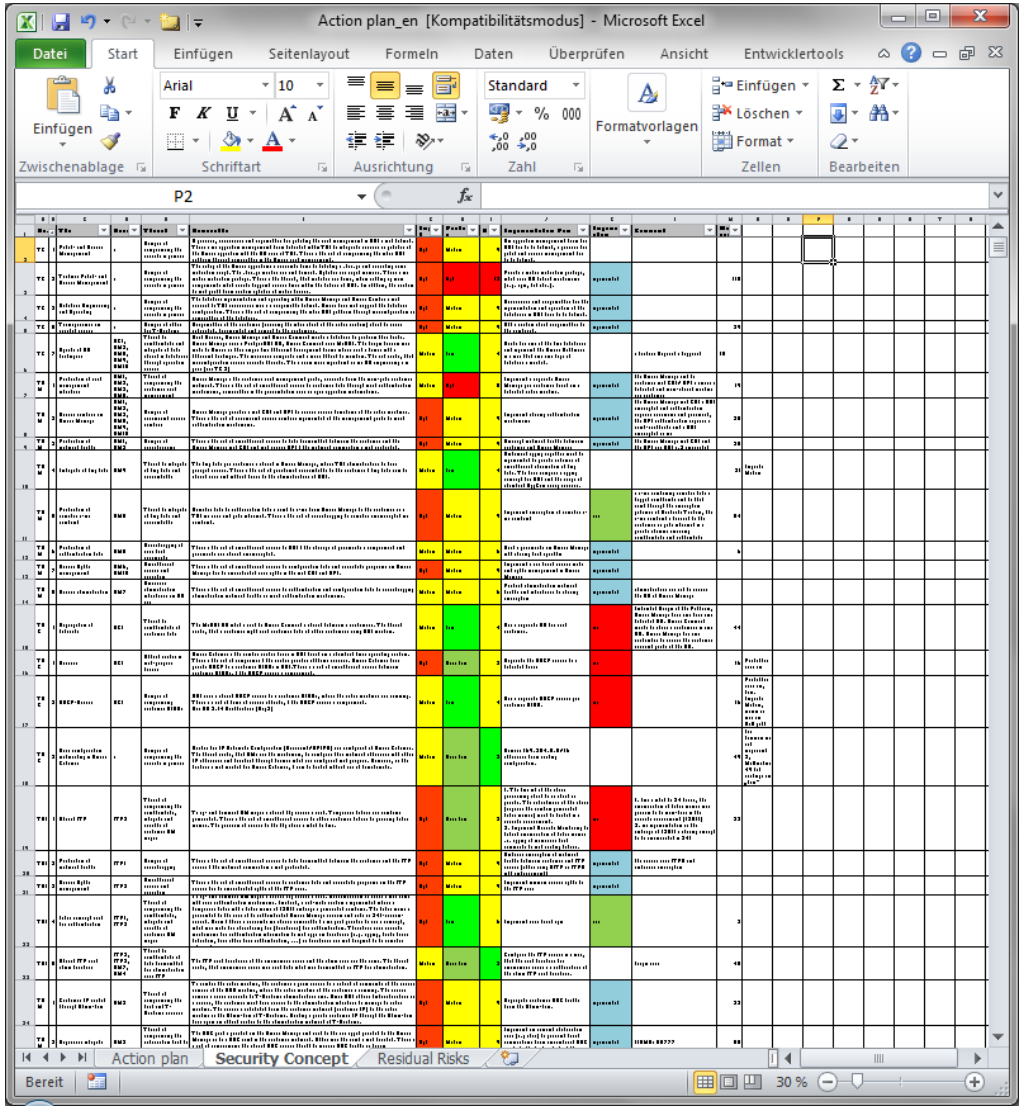
Compliance


Security-SoC [Kompatibilitätsmodus] - Microsoft Excel
Datei
Start
Einfügen
Seitenlay
Formeln
Daten
Überprüf
Ansicht
Entwickle
Arial
10
Einfügen
Zwischenablage
Schriftart
Ausrichtung
Zahl
Formatvorlagen
Zellen
Bearbeiten
I9
SR 3.14 Architecture
SR 3.21 Unix Systems
SR 3.15 Windows S
Bereit
30 %

Risikoanalyse

- Aufnahme von Requirementerfüllungen/-verletzungen
- Bewertung (Eintrittswahrscheinlichkeit/Gefährdungspotential)

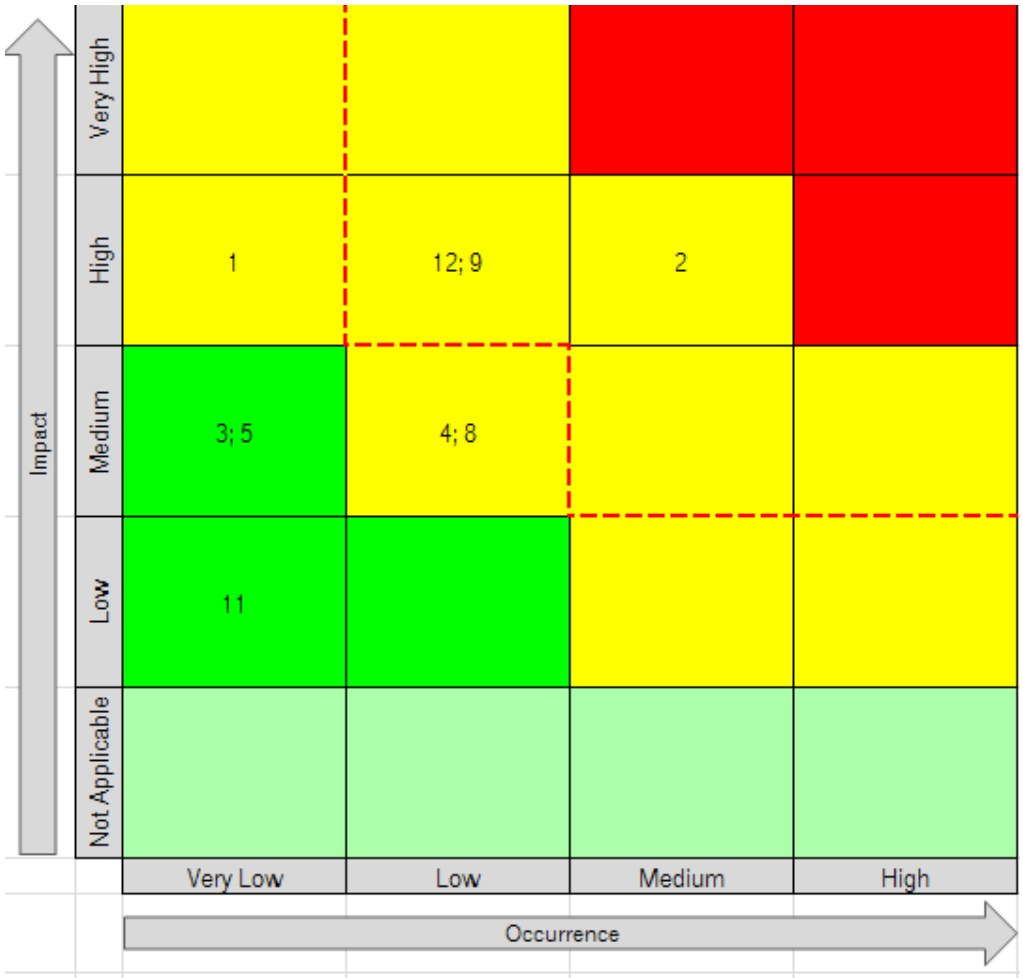
Risikoanalyse



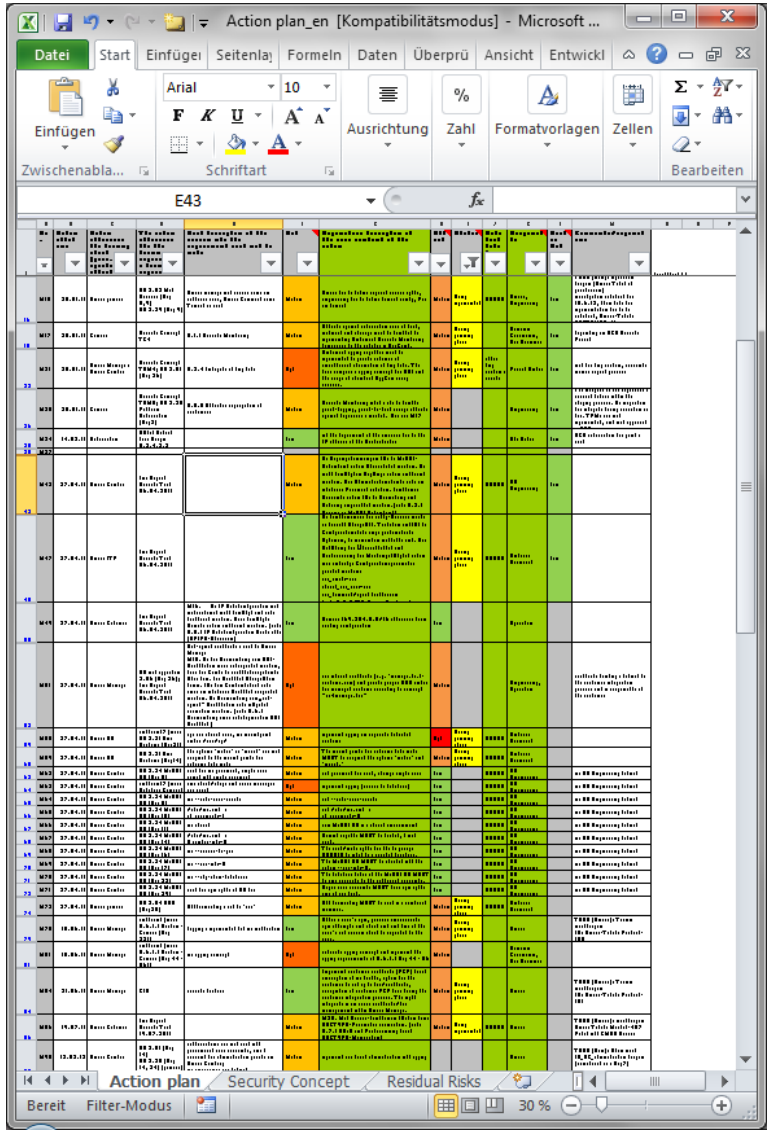
Risikoanalyse

- Aufnahme von Requirementerfüllungen/-verletzungen
- Bewertung (Eintrittswahrscheinlichkeit/Gefährdungspotential)
- Nachbesserung/Risikoakzeptanz

Akzeptanz


Impact
Very High
High
Medium
Low
Not Applicable
1
12; 9
2
3; 5
4; 8
11
Very Low
Low
Medium
High
Occurrence

Maßnahmen


Action plan_en [Kompatibilitätsmodus] - Microsoft ...
Datei
Start
Einfügen
Seitenlay
Formeln
Daten
Überprü
Ansicht
Entwickl
Arial
10
Einfügen
Zwischenabla...
Schriftart
Ausrichtung
Zahl
Formatvorlagen
Zellen
Bearbeiten
E43
Action plan
Security Concept
Residual Risks
Bereit
Filter-Modus
30 %

Probleme

- sehr lange Listen
- Grün ist einfacher als Rot
- Fragen teilweise unspezifisch

Penetrationstest

Probleme

- sehr lange Listen
- Grün ist einfacher als Rot
- Fragen teilweise unspezifisch
- Risikoabschätzung wird ausgelassen

Zusammenfassung

- Compliance $\rightarrow$ Risiko $\rightarrow$ Akzeptanz $\rightarrow$ Maßnahmen

Zusammenfassung

- Compliance → Risiko → Akzeptanz → Maßnahmen
- Google | BSI IT-Grundschutz
- `https://owasp.org`

Zusammenfassung

- Compliance → Risiko → Akzeptanz → Maßnahmen
- Google | BSI IT-Grundschutz
- `https://owasp.org`