Security-Audit von http://www.aeroclub-nrw.org/

Max Kellermann

29. April 2008

Zusammenfassung

http://www.aeroclub-nrw.org/ ist die CAMOplus-Plattform des Deutschen Aero-Club LV NRW. Es handelt sich um eine Webapplikation, die im Browser bedient wird.

Dieses Dokument ist das Ergebnis eines Security-Audit der technischen Plattform.

Inhaltsverzeichnis

1 Entstehung des Audit-Projekts

Während des CAMO-Vortrags von Ulf Calsbach auf der Fluglehrerfortbildung am 9. März 2008 auf dem Dümpel hatte ich Zugang zu der Webseite. Aus Neugier probierte ich eine sehr einfache Attacke auf die Webseite: ich gab ein Hochkomma als Usernamen ein. Es erschien eine Fehlermeldung, die sofort darauf schließen ließ, daß die Webseite per SQL Injection verwundbar ist.

Das untersuchte ich weiter, und fand das System an allen Stellen blank; alle Sicherheitsmechanismen (sofern überhaupt vorhanden) ließen sich auf einfachste Weise umgehen. Schließlich war es mir möglich, die gesamte Datenbank herunterzuladen. Dort fand ich alle Paßwörter im Klartext, ebenso wie viele sensible Daten (Adressen aller Mitglieder und ehemaligen Mitglieder aller NRW-Vereine).

2 Anforderungen an die Sicherheit

In Annex I Part M werden die Anforderungen nur sehr spärlich beschrieben:

"In case of a computer system, it should contain programme safeguards against the ability of unauthorised personnel to alter the database."

"The person issuing the certificate of release to service should use his normal signature except in the case where a computer release to service system is used. In this latter case the competent authority will need to be satisfied that only the particular person can electronically issue the release to service. One such method of compliance is the use of a magnetic or optical personal card in conjunction with a personal identity number (PIN) known only to the individual, which is keyed into the computer. A certification stamp is optional."

Das erste Zitat klingt offensichtlich, ebenso wie die Grundaussage des zweiten Zitats. Das darauf folgende Beispiel-Konzept zeigt, wie schwammig diese allgemeine Anforderung an die Sicherheit eigentlich formuliert sind.

Wie wichtig die Sicherheit wirklich ist, erschließt sich aber sofort, wenn man die Rolle der Software bedenkt: hier wird elektronisch abgewickelt, was früher die Handschrift und die Unterschrift des Prüfers erledigte. Ob eine LTA rechtzeitig durchgeführt wird, oder ob man sich blind auf die (manipulierte) Software verläßt, kann im Extremfall über Menschenleben und vermeidbare Unfälle entscheiden. Im besten Fall müßte eine unsichere CAMO wohl "nur" ihren Betrieb einstellen.

3 Vorhandene Sicherheitsmechanismen

3.1 HTTP-over-SSL

Die Verbindung zum Browser des Benutzers kann optional per SSL verschlüsselt werden.

Das Zertifikat wurde 2005 ausgestellt und hat eine Laufzeit von 5 Jahren. Die Daten im Subject des Zertifikats sind falsch; nur der fuer die automatische Verifikation nötige Common Name (CN) enthält den korrekten Wert.

Die Key Usage des Zeritifikats enthält neben "TLS Web Server Authentication" auch noch "TLS Web Client Authentication".

3.2 Login per Paßwort

Zugang zu Daten erhält man erst nach Login mit Accountname und Paßwort. Das Paßwort kann der Benutzer selber wählen. Es gibt keine Paßwort-Policy. Die Software generiert Paßwörter, die 4 Zeichen lang sind und nur aus Buchstaben und Ziffern bestehen.

3.3 Session-Handling

Nach erfolgreichem Login bekommt man eine Session-ID zugewiesen. Diese dient bei jedem weiteren Request als Ersatz für Benutzername und Paßwort. Sie wird als Request-Parameter entweder im Query-String oder im Post-Body übertragen, und muß daher auch Bestandteil jeder HTML-Seite sein. Sie bleibt bis zum Logout gültig.

3.4 Rollenbasierter Zugriff

Jedem Benutzer-Account wird eine Rolle zugewiesen. Je nach Rolle variieren die Berechtigungen des Benutzers.

4 Beschreibung der technischen Plattform

4.1 Der alte Server

Die gesamte Software läuft isoliert auf einem einzigen Server, der bei 1&1 gemietet wurde. Als Betriebssystem wurde SuSE Linux mit dem sehr alten Kernel 2.4 gewählt. Die Daten liegen in einer MySQL-Datenbank.

Als Webserver wurde Apache 1.3.34 gewählt, die SSL-Verschlüsselung übernimmt das Modul Ben-SSL 1.55.

Die Software selber wurde in PHP geschrieben. PHP 5.2.5 wird als CGI in den Webserver eingebunden. Auf Hardened PHP bzw. Suhosin wurde verzichtet.

Es ist keine Backup-Lösung im Einsatz. Im Fall eines Hardware-Ausfalls sind alle Daten verloren.

4.2 Der neue Server

Den neuen Server habe ich eingerichtet; es handelt sich um einen Root-Server bei 1&1 mit Debian Etch, aktuellstem Linux-Kernel und SELinux. Als Webserver ist Apache 2.2.3 im Einsatz, PHP Version 5.2.0 wird als CGI mit Suhosin und einer eigenen SELinux-Rolle betrieben. Die Versionen sind zwar teilweise nicht die neuesten, enthalten aber wie bei Debian üblich Patches für alle bekannten Sicherheitslücken.

Auch auf dem neuen Server wird kein Backup durchgeführt. Ein Konzept dafür habe ich Anfang April vorgelegt, es wurde jedoch bisher nicht umgesetzt.

5 Entwicklung

Die Software wurde von Wilfried Zingel und Hans Otto Edelhoff entwickelt. Sie verwenden als Werkzeug dafür die freie Entwicklungsumgebung Eclipse.

Es ist kein Werkzeug zum Source Control Management im Einsatz. Die gemeinsame Entwicklung beider Autoren findet auf einem zweiten Webserver-Host statt.

Zur Erleichterung ihrer Arbeit haben die Autoren Code-Generatoren entwickelt.

Während des Projekts wurde IT-Sicherheit nicht thematisiert. Wilfried Zingel sagte mir, es sei "keine Zeit" dafür gewesen, außerdem kenne er sich damit nicht aus. Darum habe er sich später kümmern wollen; konkret sei aber nichts geplant gewesen, auch kein Zeitpunkt.

6 Konzeptionelle Fehler

IT-Sicherheit – genau wie die Flugsicherheit – ist nichts, was man einem Produkt einfach hinzufügen kann. IT-Sicherheit passiert in den Köpfen der Menschen, die teilnehmen. Deswegen ist es wichtig, proaktiv Sicherheit zu schaffen: man versucht, potentiell gefährliche Verhaltensmuster zu vermeiden, und hält sich an bestimmte allgemein bekannte Regeln.

Diesem Prinzip sind die Entwicklern nicht nachgegangen. Es folgt eine kurze Liste ohne Anspruch auf Vollständigkeit.

6.1 Authentizität der Daten

Die Software kann nicht feststellen, wer Daten angelegt und verändert hat. Es wurde auf elektronische Signaturen verzichtet. So kann die Echtheit von Prüfereingaben nicht bewertet werden. Eine Zulassung als CAMO-Betrieb scheint mir vor dem Hintergrund fragwürdig, denn die Software kann gar keine verlässlichen Aussagen treffen, da keine Möglichkeit zur Verifizierung vorgesehen ist.

6.2 Session-ID in der URL

Die Session-ID wird in der URL übertragen. Das bedeutet, daß sie in der Browser-History und in Proxy-Logs gespeichert ist, und sie kann während einer Sitzung abgelesen und entwendet werden. Wenn man eine gültige Session-ID hat, entfällt der normale Login mit Benutzername und Paßwort.

Im "Referrer"-Header überträgt der Browser die URL der vorher besuchten Seite. Wenn man also einem Link auf eine fremde Seite folgt, überträgt man damit automatisch seine Session-ID zum Besitzer der Zielseite, der wiederum damit den CAMO-Account seines Besuchers kompromittieren kann.

Während eines Beamer-Vortrags kann man die Session-ID ebenfalls klauen, indem man ein Foto der Leinwand macht, oder die Adresszeile abtippt.

6.3 Übertragung unnötiger Daten

Mit jedem Request wird der Benutzername und der Passwort-Hash erneut übertragen. Das ist überflüssig, und führt zu einer Erweiterung des Angriffspotentials. Da auch diese Daten in der URL ablesbar sind, ist es sehr einfach, den Hash eines fremden Passworts zu bekommen, woraus sich das Paßwort zurückrechnen läßt.

6.4 Schreibzugriff

PHP hat Schreibzugriff auf alle Daten und auf den Code. Es ist eine gute Praxis, einem Prozess nur die Rechte einzuräumen, die für seine Arbeit nötig sind. So schließt man eine Möglichkeit aus, Code zu manipulieren.

6.5 PHP

PHP ist eine einfach zu lernende Programmiersprache, die speziell für Webanwendungen entworfen wurde. Sie hat sich Ende der 90er gegen das vorher dafür sehr beliebte Perl durchgesetzt. Man schreibt einfach HTML und bettet die PHP-Befehle darin ein. Aus Unternehmersicht ist PHP interessant, weil die Personalkosten für PHP-Programmierer deutlich niedriger als der Durchschnitt sind, was aber ein Trugschluß ist, wenn man Qualitätskriterien mit einbezieht: sind PHP-Programmierer billig, oder lernen vorrangig billige (weniger qualifizierte) Arbeitskräfte PHP?

Unter dem Aspekt der IT-Sicherheit spricht eine ganze Menge gegen PHP. Davon könnte man sicher ein ganzes Buch schreiben, hier nur ein paar der wichtigsten Punkte:

Wie keine andere Plattform wurde PHP für seine Sicherheitslücken berühmt. Immer wieder wurden kritische Löcher gefunden, die es ermöglichten, in jeden Server einzubrechen, auf dem PHP nur installiert war - selbst wenn es nicht benutzt wurde. Im "Month of PHP Bugs" vor einem Jahr hat der Security-Experte Stefan Esser jeden Tag nicht nur eine, sondern gleich 3 bis 5 neue Sicherheitslücken veröffentlicht, von denen das PHP-Team schon Monate vorher wußte, aber nicht gehandelt hat.

Als wenn das nicht schon schlimm genug wäre, scheint das Ziel von PHP zu sein, die Entwicklung sicherer Programme möglichst schwer zu machen:

- die Standard-Methode, um Werte (zum Beispiel aus der Datenbank oder aus Formularen) darzustellen, ist verwundbar gegen XSS (siehe weiter unten). Um sich dagegen abzusichern, muss man an jeder Stelle die Funktion `htmlentities()` aufrufen. Bei anderen Plattformen ist dieser sichere Modus Standard.

- der `include`-Befehl ist der Ansatz zur Modularisierung: Bibliotheken mit wiederverwendbarem Code kann eingebunden werden. Der Befehl kann aber nicht nur Bibliotheken laden, sondern auch lokal injizierten Code, und sogar Bibliotheken über das Netzwerk nachladen. Das ermöglicht eine Vielzahl von Attacken.

- PHP bietet Schnittstellen zu diversen Datenbanken, allerdings auf niedrigstem Level, und jeder Programmierer muß sich um die Absicherung selber kümmern. So muß man sich um das Quoten von Benutzereingaben selber kümmern. Erst mit PDO und PEAR kamen sichere Datenbankschnittstellen dazu, die aber immer noch zu selten benutzt werden.

- die Option `register_globals` ermöglicht es einem Angreifer, interne PHP-Variablen zu überschreiben. Das wird heutzutage zum Glück

meistens deaktiviert, aber die reine Existenz so einer Option belegt das mangelnde Sicherheitsbewußtsein des PHP-Teams.

- die Option `magic_quotes_qpc` vermittelt ein Gefühl von Sicherheit, die nicht vorhanden ist.
- auch die Option `safe_mode` schafft kaum Sicherheit, außerdem kann ein Angreifer sie deaktivieren.
- PHP gibt zu viel direkte Kontrolle an die Bibliotheken ab, die es verwendet. Zum Beispiel schert sich CURL nicht darum, wo es seine Daten herholt oder hinschreibt. Statt unerfahrenen Programmierern ein direktes API zur Low-Level-Bibliothek CURL anzubieten, sollte PHP eines anbieten, das standardmäßig sicher ist.
- mit Redirects kann man CURL dazu bringen, die sicherheitsrelevante `open_basedir`-Konfiguration zu ignorieren.
- zu komplizierte Vergleichs-Operatoren: PHP kennt nicht nur "=" und "==", sondern auch "===". Kaum ein Entwickler kennt den Unterschied, und wenn man mal den falschen benutzt, ist die Prüfung von Authentifizierungsdaten nutzlos. Darauf basierten in den letzten Jahren viele Sicherheitslücken.

Die meisten dieser "Standard-Fehler" wurden in der geprüften Software dann auch tatsächlich gemacht.

6.6 Existenz eines Benutzernamen

In dem versteckten Eingabefeld "passwortfalsch" kann man ablesen, warum der Login fehlgeschlagen ist: war das Paßwort falsch oder existiert der Benutzername gar nicht? Das kann für einen Angreifer eine wertvolle Information sein. Da es sich um ein verstecktes Feld handelt, liegt nahe, daß die Entwickler diese Information auch gar nicht zeigen wollten. Die richtige Methode wäre dann allerdings, diese Information auch wirklich gar nicht erst zu schicken, auch nicht als "verstecktes" Feld.

7 Sicherheitslücken

Dieser Abschnitt beschreibt konkrete Attacken, die aufgrund von Implementierungsmängeln möglich sind.

7.1 SQL Injection

Die Entwickler haben korrektes Escaping der Benutzereingaben nicht durchgeführt. PHP bietet dafür spezielle Funktionen an, die seit vielen Jahren zum Standard gehören. Statt diese zu benutzen, wird direkt auf das MySQL-API zugegriffen.

Die Auswirkungen sind unüberschaubar. Praktisch beliebig kann man Daten manipulieren, unterdrücken, erzeugen - per SQL Injection kann jeder User die Datenbank vollständig übernehmen.

Eine automatische Suche hat ergeben, daß wegen SQL Injection über 5.700 Code-Stellen neu geschrieben werden müssen.

7.2 Cross Site Scripting

Mit der SQL Injection verwandt ist das sogenannte "Cross Site Scripting" (XSS). Hierbei wird der Server selber nur indirekt kompromittiert. Dafür kann man auf diese Weise eine bestehende Session "klauen", selbst wenn die Session-ID in Cookies statt in der URL übertragen würde, indem man HTML-Code oder Scripte in die Seite einbaut.

Als Schutz dagegen bietet PHP wie jede andere Programmiersprache Funktionen an (z.B. `htmlentities()`), mit denen HTML-Steuerzeichen deaktiviert werden können. Diese muß auf *alle* Ausgaben angewendet werden. Wird dies an einer einzigen Stelle vergessen, ist die gesamte Seite verwundbar durch XSS.

Hier verhält es sich ähnlich wie bei der SQL Injection: die Entwickler haben dies Gefahr an keiner einzigen Stelle berücksichtigt. Hier hat die automatische Suche über 11.000 Stellen gefunden.

7.3 Klartextpasswörter

Ein wichtiges Prinzip bei allen Implementierungen der Paßwort-Authentifizierung ist, daß die Paßwörter nirgendwo im Klartext gespeichert werden, denn so führt eine kompromittierte Datenbank dazu, daß der Angreifer in den Besitz aller Paßwörter gelangt. Daher muß die Datenbank so angelegt werden, daß die Paßwort-Felder unbrauchbar sind.

Dieses Prinzip wurde in der Software mißachtet: es gibt ein Feld, in dem alle Paßwörter in Klartext gespeichert werden. Benutzt wird das Feld nicht (mehr), und es sollte unbedingt entfernt werden.

7.4 Unsichere Passwort-Hashes

Neben dem Klartext-Paßwort-Feld wird auch der MD5-Hash das Paßworts gespeichert. Das ist eine der Möglichkeiten, um Paßwörter nicht im Klartext speichern zu müssen: MD5 soll eine "Einweg-Funktion" sein, man kann so einen Hash nicht zurückrechnen. Wenn sich ein User einloggt, errechnet man den MD5-Hash des eingegebenen Paßworts, und vergleicht diesen mit dem MD5-Hash in der Datenbank.

Daß die Wahl ausgerechnet auf MD5 fiel, ist unglücklich, da MD5 seit mindestens 1996 als unsicher gilt, und 2003 und 2006 erfolgreiche Attacken veröffentlicht wurden. Bei SHA-1 gab es in den letzten Jahren ebenfalls große Fortschritte. Man sollte also lieber stärkere Algorithmen wie SHA-384, SHA-512 oder Poly1305-AES benutzen.

Generell gilt dieses Verfahren (unabhängig vom verwendeten Hashing-Algorithmus) als anfällig gegen Attacken mit einem Wörterbuch aus vorberechneten Paßwörtern, da ein Paßwort immer denselben Hash ergibt - man kann sich also eine Hash-zu-Paßwort-Tabelle erzeugen. Google eignet sich teilweise ganz gut dafür. Daher ist es schon seit Jahrzehnten üblich, das Paßwort mit einem zufälligen "Salt" zu verknüpfen. Die PHP-Funktion `crypt()` macht genau dies, sie wurde in der Software aber nicht benutzt.

7.5 Vermischung von Bibliotheken und Content

Die Code-Bibliotheken und Konfigurationsdateien liegen in einem vom Webserver veröffentlichten Verzeichnis. So hat der Angreifer ohne Mühe Zugriff auf den Code, weil ihm der Webserver die Daten freiwillig anbietet.

7.6 Öffentliche Uploads

Wenn man Belege hochlädt, werden die Dateien in einem öffentlichen Verzeichnis abgelegt, die der Webserver eigenständig ohne Einfluß der Software ausliefert. So kann sich jeder Besucher (selbst ohne gültigen Login) alle Belege runterladen, nicht nur diejenigen seiner eigenen Flugzeuge. Die Zustimmung zur uneingeschränkten Veröffentlichung dieser möglicherweise privaten Daten wurde nicht eingeholt.

7.7 Keine Prüfung der Eingabewerte

Beispiel: bei der Statusmeldung kann man eine negative Startzahl eingeben. Auch die Flugminutenzahl kann negativ sein.

7.8 Ausspähen und Manipulation beliebiger Dateien

Wenn man ein Flugzeug bearbeiten will, wählt man es aus der Liste aus, und klickt auf "ändern/zeigen". Wenn man das Firefox-Addon "WebDeveloper" installiert hat, kann man die Flugzeug-Liste in ein Eingabefeld umwandeln, in dem man ein beliebiges Flugzeug (zum Beispiel aus einem anderen Verein) eingeben kann. Der Server prüft nicht, ob der Benutzer auch wirklich dieses Flugzeug sehen darf. Mit diesem einfachen Trick kann man jedes beliebige Flugzeug manipulieren.

Dieser Fehler wurde in jedem von mir überprüften Modul gemacht, zum Beispiel:

- Flugzeuge ändern/zeigen
- Flugzeuge Komponenten
- Flugzeuge Belege
- Fluglehrer-Auswertung
- Fluglehrer-Datenblatt
- Fluglehrer-Gültigkeit

Vermutlich haben die Programmierer die nötigen Berechtigungsprüfungen an keiner Stelle implementiert, so daß alle Module verwundbar sind.

7.9 Interne Entwickler-Menüs

Unter bestimmten Umständen erscheint statt der normalen Navigationsleiste das interne Entwicklermenü. Dort findet man Hauptmenüs mit Namen wie "DB-Funktionen", "Timo im Test", "Ötte im Test", "Wilfried im Test".

Hier sind einige ziemlich schwerwiegende Sicherheitslücken begraben, da meist nicht geprüft wird, ob der aktuelle Benutzer ein Entwickler ist.

Einige der dort aufgeführten Programme stürzen allerdings auch direkt ab. Möglicherweise zerstören sie dabei auch noch Daten. Andere sind gar nicht mehr vorhanden.

7.10 Eigene Menüs einblenden

Wenn man einen mehrzeiligen HTTP-Header sendet, und dann folgendes in der Adreßzeile einfügt, kann man die Menüs selber bestimmen (kann je nach Installationsort leicht variieren):

```
menuename=../../../../../proc/self/environ
```

So kann man aber auch Denial-of-Service-Attacken starten, indem man riesige Dateien lädt.

Auch hier wurde mal wieder auf Prüfungen verzichtet, und gegen den Grundsatz verstoßen, keine Dateinamen vom Client zu verwenden.

7.11 Remote Code Execution

Auf dem alten Server habe ich auch eine Kopie der Webseite des AC Hagen gefunden, der sich erwartungsgemäß als genauso unsicher herausgestellt hat. Am eindrucksvollsten ist folgende Zeile:

```
<?php  include "$nav.php";  ?>
```

Hier kann man einfach in der Query-Variable "`nav`" beliebige Dateinamen oder URLs übergeben, damit der Code des Angreifers auf dem Server ausgeführt wird.

Da der Code auf demselben Server lag, kann er zum Angriff benutzt werden. Daher sollte man nur Code auf den Server hochladen, der als sicher gilt.

7.12 Clientseitige Eingabeprüfung

Die Prüfung der Benutzereingaben ist für jede Anwendung sehr wichtig. Es reicht nicht, dies clientseitig per JavaScript zu machen, da dem Client grundsätzlich nicht vertraut werden darf, schließlich kann ein Angreifer diese Prüfungen deaktivieren, indem er JavaScript abschaltet.

Sehr viele Formulare benutzen in dieser Software allerdings clientseitige Prüfung der Benutzereingaben, ohne zusätzliche serverseitige Prüfung.

Beispiele:

- Fluglehrer-Datenblatt ändern
- Anweisungen Instandhaltungsbetrieb ändern
- Statusmeldung

7.13 Datenbank-Export

Unter der folgenden Adresse kann man sich ein Backup der kompletten Datenbank herunterladen:

```
http://www.aeroclub-nrw.org/daec/dbfun/asexport.in.php
```

Daß solche internen Testfunktionen auf dem Produktivserver überhaupt vorhanden sind, ist ja schon schlimm genug, aber daß dabei nichtmal die Berechtigungen geprüft werden, ist fatal. Ich habe noch einige ähnlich gefährliche Entwickler-Funktionen gefunden.

Auf dem neuen Server gab es beim Datenbankexport noch folgende Ausgabe:

```
dump=mysqldump -udbo196090723 -ptyyzauq3 db196090723 > export.sql
```

Das ist der interne Befehl, mit dem der Export angelegt wird. Hier kann jeder Besucher der Seite das interne Datenbankpaßwort ablesen, es lautet "tyyzauq3".

7.14 Leeres MySQL-Paßwort

Nach der Installation von MySQL sollte man dem Administrations-User ein Paßwort geben. Das habe ich bei der Einrichtung des neuen Servers getan. Später stellte ich fest, daß das Paßwort wieder leer war. Auf meine Anfrage diesbezüglich hat Wilfried Zingel bis heute nicht geantwortet; ich vermute, er hat es zurückgesetzt.

7.15 Zu einfache generierte Paßwörter

Wenn die Software einem Benutzer ein neues Paßwort generiert, ist es nur 4 Zeichen lang. Ein solches Paßwort kann man mit vertretbarem Aufwand brute-forcen: eine Software probiert alle möglichen Kombinationen durch, bis das Paßwort richtig ist. Bei 100 Versuchen pro Sekunde braucht man durchschnittlich nichtmal 12 Stunden, um ein beliebiges Paßwort zu erraten.

7.16 Backdoor I

Der Code enthält eine Backdoor, mit der man ohne Login-Daten Administrator mit Vollzugriff wird: man gibt als Session-ID einen der folgenden Werte ein:

- `Oette7Tinas-22hennid*76hede5`

- `Oette7Tinas-22hennid*76hede`

Beispiel: `https://www.aeroclub-nrw.org/daec/mitglieder/verwaltung.php?a=Oette7Tinas-22hennid*76hede5` (geht aktuell nur auf Anhieb, wenn man bereits als "normaler Benutzer" eingeloggt war, aber das Cookie läßt sich ebenfalls fälschen).

Der Zweck dieser Backdoor ist unklar. Technisch notwendig ist das nicht, auch nicht während der Entwicklung einer solchen Software. Falls es nicht bösartig ist, so ist es mindestens grob fahrlässig, so etwas einzubauen.

7.17 Backdoor II

Wenn man in seinem Client ein spezielles Cookie namens `uawc1` setzt (z.B. mit dem Firefox-Addon "Add N Edit Cookies"), kann man ohne Paßwort einloggen:

```
uawc1=144,0 or pckennung='EDELHOFF',EDELHOFF
```

Jetzt nur noch folgende URL anklicken: `https://www.aeroclub-nrw.org/daec/anmeldung/login.in.php?uaw=1`

Dieses Beispiel enthält eine SQL Injection, was die Attacke vereinfacht (es geht auch ohne SQL Injection): man muß nur die Vereins-ID und den Benutzernamen kennen. Ein Paßwort ist wie gesagt nicht nötig. Die dabei gewonnene Session-ID ermöglicht den Administrator-Zugriff.

Der Zweck dieser Backdoor ist vermutlich die Login-Seite für den Startlisten-Computer am Segelflugstart. Warum dieser kein Paßwort benutzen soll, bleibt dennoch unklar. Schließlich wird eine vollwertige Session angelegt, mit der man alle anderen Funktionen der Software aufrufen kann.

8 Sonstige Mängel

8.1 Eigentümer

Sowohl die Domain `aeroclub-nrw.org` als auch der bei 1&1 gemietete Server gehören Hans Otto Edelhoff privat.

Die Rechte an der Software sind ungeklärt; auf meine direkte Frage hat Wilfried Zingel im März 2008 ausweichend geantwortet; Ulf Calsbach wußte es nicht. Wenn nichts weiter vereinbart wurde, sind einzig Wilfried Zingel und Hans Otto Edelhoff als Autoren im Besitz der Urheber- und Verwertungsrechte.

Beides birgt ein Risiko für den DAeC: er ist von Einzelpersonen abhängig. Die Domains, der Server und die Rechte an der Software sollten auf den DAeC übertragen werden, damit die Funktionstüchtigkeit der Plattform immer gewährleistet werden kann.

8.2 Datenbankstruktur

Die Datenbankstruktur widerspricht allen Designgrundsätzen:

- alle Felder sind als "`NOT NULL`" gekennzeichnet, aber für jedes Feld gibt es dann einen Default-Wert (die Zahl 0 oder ein leerer String). Der Code benutzt deswegen nicht etwa die `IS NULL`-Abfrage, um zu prüfen, ob ein Feld belegt ist, sondern testet auf die Zahl 0 oder den leeren String.
- die Datenbank ist hochgradig redundant: viele Werte werden zwischen den Tabellen aufwendig kopiert, statt sie zu verknüpfen.
- es gibt keine Foreign Keys, selbst nicht für die wenigen Tabellen, bei denen das möglich wäre. Mit Foreign Keys könnte die Datenbanken die Queries besser optimieren, und würde bei Updates jederzeit die Konsistenz garantieren. Nichtmal manuelle Konsistenzprüfungen sind vorhanden.
- Datums-Felder, bei denen die Uhrzeit nicht benötigt wird, sind dennoch als `DATETIME` deklariert.
- einige Datumsfelder sind doppelt vorhanden: einmal als `DATETIME` und einmal als Textfeld.
- in der "mitglieder"-Tabelle gibt es das feld "gebdatum1" und das Feld "alter". Das ist redundant, und deshalb fehleranfällig.
- es gibt gleich mehrere Mitglieder-Tabellen: "mitglieder", "benutzer", "nichtmitglieder", "mitglieder_vorupdate". Wenn man hier die Tabellenstruktur vereinheitlichen würde, könnte man weniger und simpleren Code schreiben.
- den Entwicklern scheinen Joins und Subqueries unbekannt zu sein: statt mit einer Abfrage alle Daten auf einmal zu holen, stellt er teilweise hunderte Abfragen, nur um eine Seite darzustellen.
- scheinbar genau deswegen werden auch zuhauf temporäre Tabellen angelegt, in denen Daten gesammelt werden, statt einfach mit einem Query gleich alle Daten zu holen.

- an mindestens einer Stelle wird eine Aggregatfunktion zusammen mit einer Gruppierung verwendet, obwohl nicht aggregiert werden soll, sondern eine bestimmte Zeile gesucht wird. Die Entwickler haben scheinbar das Prinzip der Aggregatfunktionen mißverstanden.

- einige Felder haben scheinbar willkürliche Längenbegrenzungen, zum Beispiel darf der Name eines Mitglieds nur 30 Zeichen lang sein. Das kann bei Doppelnamen schnell knapp werden. Solche engen Längenbegrenzungen sind nicht nötig, und bringen auch keine Vorteile.

- die Nomenklatur der Tabellen und der Felder ist inkonsistent. Die Art und Weise der Abkürzung und der Groß- und Kleinschreibung variiert stark. So heißt ein Feld manchmal "Bezirksnr", und manchmal "bezirk".

- für "mehrzeilige" Felder wird einfach eine willkürliche Anzahl an durchnummerierten Datenbankfeldern angelegt, statt einfach ein entsprechend längeres Textfeld zu deklarieren.

- die Fixierung auf sowas wie "Zeilen" ist ebenfalls eine Designschwäche, da eine Datenbank losgelöst von ihrer Darstellung auf dem Bildschirm entworfen werden sollte. Es ist die Aufgabe der Darstellungs-Schicht, über Dinge wie Zeilenumbruch zu entscheiden.

- Feldnamen wie "zweitesfeld", "letztesfeld", "feld3" scheinen absichtlich nichtssagend zu sein.

- die Software ist auf den LV NRW fixiert: sie identifiziert einen Verein nach Bundesland, Bezirk, Vereinsnummer (statt einfach an dessen Primary Key). Falls die Plattform mal extern angeboten werden soll, sind aufwendige Umstrukturierungen nötig. Ein allgemein gehaltenes Design statt diesem starren wäre nicht mehr Aufwand gewesen - im Gegenteil. Mit einem ID-Feld zu arbeiten wäre in jeder Hinsicht besser.

- teilweise wird bei manchen Tabellen pro Jahr eine eigene Tabellen angelegt ("start_fluege2006", "start_fluege2007", ...). Das ist schlechtes Design, denn diese Struktur hat keinen Vorteil gegenüber einer einzigen Tabelle, verkompliziert aber den Code, und verhindert bestimmte Abfragen.

- bei der Tabelle "asmerker" muß man üblicherweise gleich ein Dutzend Einzelabfragen machen, weil jede Zeile nur einen Wert enthält.

- es werden keine ENUMs verwendet, sondern ein undokumentierter Buchstabe speichert den Wert.
- es gibt einige leere Tabellen, deren Sinn sich mir noch nicht erschlossen hat, zum Beispiel “berechtigung_benutzer”, “camo_einheiten”.

Das alles macht den Code unnötig komplex, fehleranfällig, langsam und unsicher.

8.3 Alter Quellcode

Im Quellcode finden sich Unmengen von altem, nicht mehr benutztem Code. Das ist nicht nur ein Zeichen dafür, daß die Entwickler ihren eigenen Code nicht im Griff haben (mit Folgen für die Qualität der Software), sondern das kann auch zu Sicherheitsproblemen führen.

Der Audit war unter anderem deswegen sehr aufwendig - es lagen überall verstreut mehrere Versionen der Unterprogramme herum, so daß man sehr schnell die Übersicht verloren hat. Das heißt natürlich auch, daß man bereits behobene Sicherheitslücken immer noch ausnutzen kann, weil vielleicht zusätzlich noch eine alte Version irgendwo auf dem Server liegt, die verwundbar ist.

Auf dem Webserver liegen gleich mehrere Kopien der Software durcheinander, plus noch ein paar externe Programme (zum Beispiel phpMyAdmin, MediaWiki). Sicherheitslücken oder Konfigurationsfehler in diesen Programmen führen ebenfalls zur Kompromittierung des ganzen Servers.

8.4 JavaScript

Für die Haupt-Navigation wird ein Browser mit JavaScript benötigt. Diese Anforderung stellt eine zu hohe Anforderung für einige Anwender dar. So haben die meisten Mobil-Browser in Handys und PDAs kein JavaScript.

Oft wird aus Sicherheitsgründen empfohlen, in Firefox oder im Internet Explorer JavaScript zu deaktivieren - dieser Empfehlung kann ich mich voll anschließen. Die NoScript-Extension für Firefox deaktiviert JavaScript standardmäßig. Man schließt also Benutzer aus, die besonderen Wert auf die Sicherheit ihres Computers legen. So zwingen die Entwickler dem Benutzer ihre eigenen unsicheren Arbeitsmethoden auf. Das halte ich für inakzeptabel.

Hier JavaScript vorauszusetzen ist also nicht nur ignorant gegenüber bestimmten Benutzergruppen, sondern auch unnötig und überflüssig. Es gibt einfache Techniken, mit denen man standardkonform und portabel bleibt, aber trotzdem die Seite optional mit JavaScript anreichern kann, falls ein Benutzer sich dafür entscheidet.

Am Telefon konnte mir Wilfried Zingel auch keinen einzigen technischen Vorteil seines JavaScript-Menüs gegenüber einen reinen HTML-Menü nennen.

Für problematisch halte ich den übermäßigen Einsatz von JavaScript auch für Behinderte: Seiten, die ohne JavaScript nicht gehen, können von vielen Behinderten nicht bedient werden. Diese Art der Diskriminierung ist unnötig, und sollte vom DAeC nicht betrieben werden.

8.5 Globale Variablen

Die Entwickler haben sich viel Mühe gegeben, möglichst auf lokale Variablen zu verzichten - so habe ich über 400 Quellcode-Dateien gefunden, die sich mit nichts anderem beschäftigen, als globale Variablen anzulegen.

Globale Variablen gelten als fehlerträchtig. Man verliert schnell den Überblick, wo man welche Variable wie gesetzt hat. Der Code wird auch sehr schwer les- und wartbar, und die natürliche Folge ist Instabilität und Unsicherheit.

In sauberem Code benutzt man nur wenige oder am besten gar keine globalen Variablen, sondern legt sich in jedem Unterprogramm nur die Variablen an, die man gerade braucht. Bei Schnittstellen übergibt man die nötigen Werte als Parameter oder als Funktions-Rückgabewerte. So ist die Lebensdauer und die Zuständigkeit jeder Variable klar geregelt; es kann keine Überschneidungen und daraus resultierende Fehlfunktionen und Sicherheitslücken geben.

8.6 Internationalisierung

Die Software unternimmt an ein paar Stellen den Versuch, international zu wirken: einige Texte werden sowohl in Deutsch als auch in Englisch dargestellt. Das sind aber nur sehr wenige. Das System, alles zweisprachig anzuzeigen, funktioniert aber nicht, denn das macht die Seite nur unübersichtlich, wenn jede Meldung in vielen verschiedenen Sprachen auf einmal erscheint.

Das wirkt ziemlich unprofessionell und inkonsistent. Dabei gibt es keinen Grund, mehrere Sprachversionen auf einmal zeigen. Man zeigt einfach nur die Sprache an, die der Benutzer auch lesen kann. Das kann man dem Request-Header `Accept-Language` entnehmen.

Beim Datum beharrt die Software auf deutschem Format, und wandelt dies intern ins USA-Format um (Funktion `datumtousa()`).

8.7 Optimiert für Firefox 2.0

Dieser Spruch stammt so ähnlich aus den 90ern, als Netscape und Microsoft sich gegenseitig im Monatstakt mit neuen Features im Browser übertrumpft haben. Heutzutage ist das alles standardisiert und ruhiger geworden. Man entwickelt nicht für einen bestimmten Browser, sondern für einen Standard (HTML 4.0, XHTML 1.0, CSS 2, ...). Wenn man dabei standardkonform bleibt, läuft das in jedem standardkonformen Browser, egal ob das nun Internet Explorer, Firefox, Safari, Mozilla, Opera, Dillo, Lynx, W3M, Amaya, ein Mobiltelefon oder gar eine Spielkonsole ist.

Das Ziel muß sein, daß man dem Benutzer die Freiheit zur Wahl seines Browsers läßt.

8.8 Feste Nummern

Bestimmte Dinge sind im Quellcode "hart verdrahtet". Obwohl in der Datenbank großer Aufwand betrieben wird, um zu speichern, wer was machen darf, gibt es dennoch ein paar feste willkürliche Zahlen:

```
if ($rowv2->vereinsid == '274'):
  $lbauser = true;
  asmerkerschreiben("lbauser",1);
endif;

if ($rowv2->vereinsid == '314'):
  $camopruefuser = true;
  asmerkerschreiben("camopruefuser",1);
endif;
```

Scheinbar gibt es ein 17. Bundesland – oder wurde das Bundesland-Datenbankfeld einfach nur zweckentfremdet?

```
if ($rowv2->Bundesland == 'LBA'):
  $lbauser = true;
  asmerkerschreiben("lbauser",1);
endif;
```

8.9 SSL-Zertifikat

SSL/TLS bietet nicht nur Verschlüsselung der Verbindung, sondern auch Authentifizierung: der Server zeigt dem Client sein Zertifikat, das von einer vertrauenswürdigen "Certificate Authority" (CA) digital unterschrieben wurde.

Die CA bestätigt dabei nicht, daß der Inhaber des Zertifikats selber auch vertrauenswürdig ist, sondern nur, daß es sich wirklich um dejenigen handelt, als der er sich ausgibt. Das bedeutet, daß das Vertrauen des Anwenders gegenüber dem Webseitenbetreiber nicht aus der Existenz eines Zertifikats resultieren kann, sondern er muß den Webseitenbetreiber bereits kennen.

Die Identität des Webseitenbetreibers kann er aus dem "Subject" des Zertifikats ablesen. Dort steht, welcher Organisation das Zertifikat gehört.

Beim Zertifikat von `www.aeroclub-nrw.org` steht dort stattdessen allerdings einfach nur "www.aeroclub-nrw.org". Richtig müsste es aber "Deutscher Aeroclub LV NRW e.V." lauten. Der achtsame Benutzer kann also nicht feststellen, daß er wirklich die Webseite des DAeC vor sich hat, sondern das kann irgendjemand weltweit sein, der zufällig diese Domain registriert hat. Damit ist das Ziel, das mit SSL erreicht werden sollte, verfehlt.

Erschwerend kommt hinzu, daß die Domain `aeroclub-nrw.org` gar nicht dem DAeC gehört. Sie befindet sich im Privatbesitz von Hans-Otto Edelhoff.

9 Zusammenfassung

Zusammenfassend kann ich nur das Urteil abgeben, daß "Sicherheit" in keiner Form vorhanden ist. Die geringen Anforderungen, die Annex I Part M stellt, werden nicht erfüllt, egal, wie weit man die schwammigen Formulierungen dehnt. Eine gründliche Überarbeitung und eine Neuentwicklung großer Teile der Software ist unumgänglich, weil die Sicherheitsprobleme sehr tief liegen.

10 Empfehlung

10.1 Schulung

Ich habe weiter oben geschrieben, daß Sicherheit in den Köpfen stattfindet. Daß die Sicherheit der Software so katastrophal ist, ist in erster Linie ein Kompetenzproblem. Hier muß angesetzt werden, und ein Lernprozess gestartet werden.

Die Ausrede "es war einfach keine Zeit für Sicherheit" ist faul und darf niemals akzeptiert werden. Man stelle sich mal einen Flugzeugkonstrukteur vor, der ein absolut unsicheres Flugzeug auf den Markt bringt, das jederzeit bei den Erstkunden auseinanderfallen kann, für die Sicherheit hatte er nämlich einfach keine Zeit, aber da will er sich jetzt drum kümmern, wofür gibt es denn LTAs... Sowas würde man mit Recht als Pfusch bezeichnen.

10.2 Neuentwicklung des gesamten Datenbank-Codes

Praktisch jede Code-Zeile, die sich mit dem Datenbankzugriff beschäftigt, enthält kritische Sicherheitslücken. Um eine Neuentwicklung wird man nicht herumkommen.

10.3 Neustrukturierung der Datenbank

Aufrgrund von weiter oben beschriebenen Mängeln sollte die Datenbankstruktur überarbeitet werden.

10.4 Verwendung einer Template-Sprache

Die Software hat zwar schon so etwas wie eine selbstgemachte Template-Sprache, diese wird jedoch nur sporadisch benutzt, und ist ebenso unsicher wie der restliche Code.

Eigentlich ist PHP selber schon eine Art Template-Sprache, die allerdings aus Sicherheitsaspekten nicht akzeptabel ist.

Es gibt viele gute Template-Sprache, deren Eignung und Sicherheit man evaluieren sollte, und die man flächendeckend und konsequent benutzen sollte.

10.5 Trennung von Datenhaltung und Webserver

Der Datenbankserver als zentrale Datenhaltung sollte physikalisch vom Webserver getrennt sein, damit ein kompromittierter Webserver nicht gleichzeitig eine administrativ kompromittierte Datenbank bedeutet.

10.6 Qualitätsmanagement

Es gibt viele Techniken zur Qualitätssicherung. Meines Erachtens das wichtigste sind automatische Unit-Tests: man schreibt für jedes Unterprogramm ein kleines Testprogramm, welches alle Funktionen testet. So kann man jederzeit die Lauffähigkeit der gesamten Plattform bewerten. Die Erfahrung hat gezeigt, daß man damit sogar schon kurzfristig viel Zeit sparen kann, obwohl die Erstellung dieser Unit-Tests selber Zeit kostet.

10.7 Source Control Management

Man sollte ein SCM wie Subversion benutzen. So kann man jederzeit alle Änderungen nachvollziehen. Diese Empfehlung ist stark verknüpft mit dem vorherigen Punkt “Qualitätsmanagement”.

10.8 Abkehr von PHP

Über die Nicht-Eignung von PHP für sichere Applikationen habe ich weiter oben schon geschrieben, daher schlage ich einen Plattformwechsel vor. Es gibt viele gute Plattformen, und meine persönliche Empfehlung ist TurboGears. TurboGears integriert diverse sichere und stabile Bibliotheken:

- *Cherry* ist der Webserver, der auch per `mod_python` in einen bestehenden Apache integriert werden kann.
- *Kid* ist eine Template-Sprache, bei der das korrekte Quoting der Standard ist. Wenn man wirklich HTML-Code aus Benutzereingaben einfügen möchte, muß man Spezialfunktionen aufrufen. Cross Site Scripting ist damit praktisch ausgeschlossen. Kid-Templates können (im Gegensatz zu PHP und den meisten Template-Sprachen) mit einem beliebigen HTML-Editor bearbeitet werden.
- *SQLAlchemy* ist ein sogenannter "Objekt-Relational-Mapper", mit dem man objektorientiert auf die Datenbank zugreifen kann. Das beseitigt die fehlerträchtige Vermischung von Code mit SQL.
- *formencode* prüft die Gültigkeit von Benutzereingaben.
- *gettext* ermöglicht die einfache Internationalisierung: alle übersetzbaren Meldungen werden automatisch gesammelt, und ein Übersetzer kann die Meldungen anschließend mit einem eigenen Programm übersetzen. Lokalisierte Datums- und Zahlenformatierung ist dabei selbstverständlich.
- mit an Bord ist alles was man sonst noch braucht: Request Routing, Authentifizierung, JavaScript, Widgets, ...

Die Entwicklung mit TurboGears ist sehr einfach und extrem schnell. Viele der Fehler in der aktuellen Software wären gar nicht erst möglich gewsen.

Zwar muß man dafür den gesamten Code neu schreiben, aber das ist meines Erachtens aufgrund der Menge und Schwere der gefundenen Mängel ohnehin Pflicht.

10.9 Umstieg auf PostgreSQL

Die Datenbank MySQL setzt ihren Schwerpunkt auf Geschwindigkeit, während das Ziel Nummer 1 bei PostgreSQL die Zuverlässigkeit ist. Das kann man an vielen Stellen spüren (wobei PostgreSQL unter hoher Last trotzdem schneller ist). Beide Datenbanken sind frei und kostenlos.

Da die Konsistenz der Daten sehr wichtig ist, schlage ich vor, auf PostgreSQL umzusteigen. Sobald der Datenbank-Code neu geschrieben ist, sollte die Umstellung kaum Aufwand sein.

10.10 Backup

Es muß ein Backup-System geben. Der Hoster 1&1 bietet ein Backup per FTP an.

Allerdings: "All computer hardware used to ensure backup shall be stored in a different location from that containing the working data in an environment that ensures they remain in good condition." (Annex I Part M)

Das schließt ein Backup bei 1&1 aus. Denkbar wäre Replikation der Datenbank zu einem anderen Ort.

11 Umsetzung

Während des Audit hat sich schon einiges getan:

- ich habe Kontakt mit dem Entwickler Wilfried Zingel, der im März die Absicht ausgedrückt hat, alle von mir gefundenen Mängel zu beheben. Im April hat er es sich allerdings schon wieder anders überlegt: bei einigen Mängeln weigert er sich schlicht, diese als Sicherheitslücken anzuerkennen, siehe weiter unten.

- die Plattform wurde auf einen neuen Server umgezogen, den ich eingerichtet habe. Dabei habe ich auch SELinux installiert. Das ist ein System, das versucht, den Schaden durch unsichere Software zu begrenzen. Das ist wie ein Fallschirm: man darf trotzdem nicht unsicher fliegen oder programmieren.

11.1 Der PDO-Trugschluß

Wilfried Zingel hat angefangen, die sichere Datenbankschnittstelle "PDO" zu benutzen, allerdings hat er dies auf eine Weise getan, die alle alten Sicherheitslücken weiterhin zuläßt. Er beharrt jedoch darauf, daß der Code nun sicher ist, und arbeitet angeblich an einem Beweis für diese abenteuerliche These (auch nach 2 Wochen ist noch kein Beweis geliefert worden).

Diese These ist nicht nur falsch, sondern der Beweis auch praktisch unmöglich - ein Beweis für die Sicherheit einer Software wäre mathematisch zwar möglich,

aber aufgrund der extrem hohen Komplexität für Menschen nicht praktikabel. Der Mangel eines bekannten Sicherheitslochs ist kein ausreichender Beweis für die Sicherheit! Diesem (logisch falschen) Trugschluß verfallen leider viele Menschen.

Hier mein Beweis, daß Wilfried Unrecht hat, und sein Code immer noch unsicher ist:

Alter Code (Zitat aus `allgemein/asfunktionen.php`):

```
$sql = "select * from anmeldungen where name = '$anmeldung'";
```

Die neue und angeblich sichere Variante:

```
$sql = "select * from anmeldungen where name = :name = ''$anmeldung''";
```

Der Denkfehler hier ist, daß zwar PDO benutzt wird um die Werte an die Datenbank zu übergeben, der Befehl jedoch bereits vorher kompromittiert sein kann. Der (manipulierte) Befehl läuft also durch einen neuen Filter, der die Sicherheitsprobleme der alten Variante in die eigentlich sichere PDO-Variante überträgt.

Der Angriff auf den alten Coden könnte so aussehen:

```
anmeldung=' or ''='
```

Der SQL-Befehl kommt dann so bei MySQL an:

```
select * from anmeldungen where name = '' or ''=''
```

Beim der angeblich sicheren neuen Variante läßt sich der gleiche Angriff folgendermaßen durchführen:

```
anmeldung='' or 1=1 or name = :name2 = ''
```

Die Funktion `machsql()` macht daraus folgendes:

```
select * from anmeldungen where name = :name or 1=1 or name = :name2
```

Da man nun mit doppelten Hochkommas arbeiten muß, die in SQL nicht gültig sind, muß man den Exploit etwas umständlicher gestalten, was aber nichts an seiner technischen Machbarkeit ändert.

11.2 Aktueller Stand

Es hat bisher **keine** Fortschritte zur Absicherung der Software gegeben! (Stand 28. April 2008)

11.3 Aussicht

Eine Überarbeitung des vorhandenen Code wird sehr viel Zeit in Anspruch nehmen, und ich schätze, daß das bis zum Stichtag 28. September 2008 völlig unmöglich ist. Schließlich haben die Entwickler auch nach fast 2 Monaten noch nicht damit angefangen, und es sind nichtmal mehr 5 Monate übrig. Bis Ende des Jahres werden sie es bestimmt nichtmal schaffen, nur die SQL-Injections zu beheben.

Ich halte es dagegen sehr wohl für möglich, bis dahin auf Basis von TurboGears die wichtigen Funktionen neu zu entwickeln. In einem Experiment dazu habe ich innerhalb von nur zwei Tagen schon beachtliche Ergebnisse erzielt:

`http://test.camoplus.org/`

Diese Software biete ich dem DAeC NRW unter den Bedingungen der GNU General Public License Version 2 an. Dabei handelt es sich um eine Lizenz für freie Software: der DAeC darf das Programm kostenlos benutzen, kopieren, weitergeben und weiterentwickeln, ohne die bisherigen Urheber um Erlaubnis bitten zu müssen.

Mein Ziel ist es dabei, einen alternativen Vorschlag für die Umsetzung der CAMO zu liefern, der viele Probleme der aktuellen Software löst:

- alle in diesem Dokument beschriebenen Sicherheitsprobleme gibt es nicht; das ist natürlich keine Garantie für absolute Sicherheit, aber für relativ gesehen mehr Sicherheit.
- durch die klare Lizenz sind urheberrechtliche Probleme und Rechtsunsicherheiten wie bei der aktuellen Software sind damit ausgeschlossen. Es gibt keine juristischen Abhängigkeiten von Einzelpersonen, wie bisher.
- die Weiterentwicklung geht einfacher, schneller und sicherer als mit PHP.

Optimalerweise kann die neue Software bald statt der alten PHP-Software eingesetzt werden.