F A C H H O C H S C H U L E W E D E L

S E M I N A R A R B E I T

in der Fachrichtung

Medieninformatik

Thema:

Build It, Break It, Fix It:
Contesting Secure Development

Macht öffentliche Sicherheitsentwicklung mehr Sinn als geheime?

Eingereicht von:	Leon-Maxim Meyer Königsbergstraße 7 22880 Wedel Tel. 01729759669
Erarbeitet im:	7. Semester
Abgegeben am:	20. September 2018
Referent (FH Wedel):	Prof. Dr. Gerd Beuster Fachhochschule Wedel Feldstraße 143 22880 Wedel Tel. (0 41 03) 80 48-38

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Der steigende öffentliche Drang zu mehr Transparenz und Mitspracherecht, sowie die immer weiter wachsende Vernetzung der Menschen rückt das Thema „Cyber-Security“ immer mehr in den Mittelpunkt. Viele sind verschreckt von der Komplexität von Datenschutzrichtlinien [15] oder Gesetzen und verunsichert durch stetige Meldungen von Datenlecks bei Internetgiganten wie Facebook [17] und Apple [18] oder aber auch den Spionageaktionen von Regierungen [16].

Die Entwicklung der Gesellschaft hin zum Internet 4.0 mit verbreiteter Cloud- und IAC-Technologie[1] verlangt einen neuen höheren Sicherheitsstandard [14] für welchen es mehrere aktuelle Ansätze gibt. Einerseits eine firmen- oder regierungsgestützte [20], geheime Lösung, in der niemand wirklich weiß, was genau passiert und was mit den Daten geschieht, die einfach zugänglich ist und weit verbreitet.[23] Oder andererseits eine OpenSource Lösung [19], welche es dem Nutzer erlaubt, die Arbeitsweise und die Datenverarbeitung einzusehen und selbst zu bestimmen. Diese Lösungen sind meist unbekannter und Linux basiert.

2. Problemidentifikation

2.1 Security vs. Safety

Beide Begriffe bedeuten im Deutschen Sicherheit, sinngemäß übersetzt stehen sie aber für völlig unterschiedliche Dinge. Der eine Begriff, Security, steht für die Sicherheit eines Objektes gegen Angriffe oder Einflüsse von außerhalb. Safety hingegen steht für Sicherung der Software gegen interne Fehler, also auch die Möglichkeit von innen heraus Fehler zu behandeln und zu verhindern. Ein guter Vergleich ist eine Brandschutztür: Im Blick auf Security sollte sie nicht vorhanden sein, um Einbrechern keine Chance zu geben, im Blick auf Safety jedoch sollte diese Tür immer offenstehen, um im Brandfall ein schnelles Entkommen zu ermöglichen. Diesen Widerspruch unter einen Hut zu bekommen ist die Essenz eines gut funktionierenden und sicheren Programms.[11][12]

1 IAC = Interapplication Technologie

2.2 Secure Development

Um sich mit der Frage nach der Art der Sicherheitsentwicklung auseinander zu setzen, sollte man zuerst klarstellen, was Sicherheitsentwicklung überhaupt ist.

Das Konzept von Microsoft, das im Jahre 2004 veröffentlicht wurde, kann hier einen klaren Überblick verschaffen.

Folgende Grundsätze wurden gewählt:

- Secure by design
  Schon in der Planungsphase sollte auf die Sicherheitsbelange der Software eingegangen werden.
- Secure by default
  Trotz sorgfältigster Planung sollte ein Entwickler von dem Vorhandensein von Sicherheitslücken ausgehen. Aus diesem Grund sollten die Standardeinstellungen (z.B. erforderliche Privilegien) möglichst niedrig gewählt werden und selten benutzte Features standardmäßig deaktiviert werden.
- Secure in deployment
  Die mitgelieferten Dokumentationen und Tools sollen die Administratoren dabei unterstützen, die Software möglichst optimal einzurichten.
- Communications (Software)
  Die Entwickler sollten offen mit möglichen Sicherheitslücken umgehen und den Endanwendern schnell Patches oder Workarounds zur Verfügung stellen.
- Privacy by design
  Schon in der Planungsphase sollten Datenschutzbelange der Software berücksichtigt werden.
- Privacy by default
  Die Standardeinstellungen der Software sollten konservativ gewählt werden.
- Privacy in deployment
  Datenschutzmechanismen sollten offengelegt werden, um es Administratoren zu ermöglichen, die internen Datenschutzrichtlinien des Unternehmens umzusetzen.
- Communications (Privacy)
  Datenschutzerklärungen sollten transparent formuliert werden. Ein Team für Datenschutzvorfälle sollte eingerichtet werden. [21]

Diese Grundsätze bilden stark vereinfacht eine Grundlage für eine erfolgreiche Entwicklung eines sicheren Systems.

Um aber effektiv gut zu programmieren gibt Microsoft auch ein Ablaufmodell an, welches wie folgt aussieht:

1. Anforderungsphase: Definition von Anforderung und Umgebung
2. Entwurfsphase: Identifikation der Komponenten und Umstände
3. Implementierung: Bau des Projekts
4. Überprüfungsphase: Test des Projekts anhand der Anforderungen
5. Veröffentlichung: Release
6. Reaktion auf Fehler und Probleme [21]

Werden diese Punkte nacheinander abgearbeitet, ist aus Planungssicht jede Problemzone einmal adressiert worden und möglicherweise dadurch grobe Fehler verhindert.

3. Lage der Entwicklung

3.1 Historie

In der Vergangenheit begann das Thema der IT-Sicherheit mit einem leichten herantasten an die Möglichkeiten. Erste Versuche wurden zur Belustigung genutzt und ohne kriminellen Hintergedanken. Die erste Aufzeichnung eines Sicherheitsrisikos, Anfang 1970, war über den „Creeper“, ein Programm, welches sich über das Arpnet verbreitet hat und neckische Nachrichten hinterließ. Als Resultat darauf gab es auch direkt das erste „Jäger“-Programm, den „Reaper“, welches versuchte den „Creeper“ zu fangen und zu eliminieren. Dies war sozusagen, dass erste Programm das als Antivirusprogramm diente.[13]

Erst Jahre später hat der deutsche Hacker Markus Hess durch seine Arbeit für den KGB aufgezeigt, wie anfällig die Netzwerke und Rechner sind, indem er Sicherheitslücken in einem Mailprogramm ausnutze um sensible Daten an die Sowjetunion zu verkaufen.[22] Daraufhin kam es zu einem Umdenken und die Umstellung startete, dass Sicherheit nicht mehr eine spaßige Angelegenheit ist, sondern ein ernst zu nehmendes Problem. [10]

3.2 Gegenwart

Heute sind die Risiken nicht mehr zur Belustigung. Fehler sind von kritischer Natur und können nicht mehr toleriert werden. Durch die Vernetzung vieler Systeme müssen nicht mehr nur ein paar Angriffspunkte geschützt, sondern es muss im vornherein dafür gesorgt werden, dass so wenig Angriffspunkte wie möglich entstehen. Auch bei der Entwicklung muss von Anfang an damit gerechnet werden, dass die Attacken, wie sie schon in der Vergangenheit passiert sind, erneut geschehen. Mittlerweile ist die Gesellschaft und die Entwicklung soweit vorangeschritten, dass Systeme so gut geschützt sind, sodass es wieder der Mensch ist, welcher die größte Fehleranfälligkeit besitzt.

3.3 Zukunft

Die Gegner und Gefahren eines sicheren Systems werden sich immer weiter entwickeln, stärker und gefährlicher werden, jedoch aber auch die andere Seite, die Menschen und Sicherheitssysteme, werden sensibler, besser geschult und vorsichtiger. Steigende Zahlen von vernetzen Geräten sorgen für einen immer riesiger werdendes Netz von Möglichkeiten. Die gegenwärtige Lage erlaubt es, Schritte auch in andere Richtungen zu gehen. Machine Learning und KI´s werden helfen, den Menschen zu unterstützen und Gefahren selbständig zu erkennen und zu beseitigen. [9]

4. Geheime Entwicklung

4.1 Idee

Die Idee hinter einer geheimen oder auch geschlossenen Entwicklung ist die, um durch Verschleierung von Quellcode und Schnittstellen, eine Art Schutz des Codes und der Strategien zu ermöglichen um dadurch eine Vervielfältigung oder die Einsicht in den Code zu verhindern. [6]

4.2 Vorteile

Der Vorteil von geheimer Entwicklung ist vorrangig der, dass die Konfektion und die Konfiguration von Soft- oder Hardware nicht einfach zugänglich ist. Angreifer müssten als aller erstes versuchen mögliche Schwachstellen ausfindig zu machen, um diese dann auszunutzen.[24] Außerdem wird dadurch eine Art Unantastbarkeit simuliert, die abschreckend wirkt.

Entstehen Fehler oder Sicherheitslücken, können Reichweite und Schwere unter Verschluss gehalten werden und werden nicht in der Öffentlichkeit bekannt, sofern diese nicht von erheblicher Bedeutung sind, wie bei Facebook oder Apple. [25]

Wirtschaftlich gesehen, ist diese Methode in der Entwicklung deutlich teurer, da jeder Teil selber neu entwickelt werden muss, jedoch muss danach weniger Geld für Instandhaltung ausgegeben werden. Ein weiterer Vorteil aus wirtschaftlicher Sicht ist eine Art Monopolstellung für die Soft- oder Hardware, weil niemand anderes außerhalb des Unternehmens Kenntnisse über das Produkt hat. [26]

4.3 Nachteile

Die Nachteile sind hier auch nicht zu vernachlässigen. Wenn man die Vorteile von einem geringen Datenfluss nach außen hin nutzen will, muss dies natürlich auch für Mitarbeiter und Eingeweihte im Unternehmen gelten. Die Gefahr durch „Whistleblowern“, wie Edward Snowden ist imminent und muss durch gute Unternehmensführung und Mitarbeiterbehandlung verringert werden.

Zu der auch als Vorteil genannten eigenständigen Fehlerbehandlung und Monopolstellung bezüglich der Software kommt die alleinige Verantwortung gegenüber den Kunden zum Tragen, woraus riesige Skandale und Kosten entstehen können. Des Weiteren sorgt die interne Entwicklung dafür, dass es immer nur eine begrenzte Augenzahl gibt, die das Produkt überwacht und testet. Mögliche Fehler können dadurch übersehen werden und zu Gefahren werden. [27]

Da der Mensch immer dazu geneigt ist, immer Neues zu entdecken und auch gerade Hacker immer gewillt sind, das Unmögliche zu schaffen, sind gerade Produkte die einen unbekannten Faktor enthalten, im Mittelpunkt des Interesses solcher Leute. Dies kann dazu führen, dass der Vorteil der Abschreckung und des unwissenden Gegners als erstes fällt oder als Angriffspunkt dient. [28]

4.4 Beispiele

Als Beispiel sind momentan fast alle Programme genannt, die eine Art Verschlüsselung ihres Quellcodes besitzen oder die Einsicht und mögliches Reverse-Engineering verhindern wollen. Genauso aber auch Software, bei welcher der Nutzer nur die Nutzungsmöglichkeit der Software und keine Kopie der Software bekommt.

Beispiele sind: Microsoft Windows, iTunes, Adobe Photoshop, macOS u.v.m..[6]

5. Öffentliche Entwicklung

5.1 Idee

Die Idee hinter einer öffentlichen Entwicklung, also einer Art OpenSource ist die, dass alle Nutzer des Produktes den Quellcode einsehen können und dadurch volle Übersicht über das Produkt erlangen können. Außerdem ist es erwünscht, dass jeder Nutzer Quellcode ändern und erweitern darf, um so ein großes Konvolut von transparenten Schnittstellen und Programmen zu schaffen.

5.2 Vorteile

Ein riesiger Vorteil der öffentlichen Entwicklungsidee ist klar: Die Transparenz, die dadurch entsteht, dass viele Augen von vielen Mitwirkenden sehr viel Varianz aber auch Kontrolle in das Projekt bringen. Dadurch gibt es nicht nur eine Vielzahl an möglichen Lösungen, sondern auch eine Art Sicherheit gegenüber des bereits verwendetem Codes. Wird nämlich eine Sicherheitslücke offenbart, suchen sofort mehrere verschiedene Menschen nach einer Lösung, um diese schnell zu veröffentlichen und weiterhin für jeden Nutzer dieser Idee ein sicheres System zu bieten.

Die Masse an Menschen ist auch ein weiterer Vorteil, da durch die weltweite Verbreitung der veröffentlichte Code leicht verständlich und gut reproduzierbar sein muss.

Aus wirtschaftlicher Sicht kann OpenSource auch sinnvoll sein, da es nicht mehr notwendig ist, einen neuen eigenen Code zu produzieren sondern nur noch einen schon vorhandenen Code zu modifizieren wodurch sich Unternehmen Entwicklungs- und Instandhaltungskosten sparen. [8]

5.3 Nachteile

Die Nachteile hierbei sind nicht zu verhindern.

Ist es in einem Projekt notwendig, eine speziell zugeschnittene Lösung zu erreichen, kann es nötig sein, komplizierte öffentliche Lösungen zu nutzen, statt einfache kurze Programme, nicht öffentlicher Natur, selber zu entwickeln und zu nutzen. Die Open-Source-Lösungen müssen dann kompliziert umgeschrieben werden und machen dadurch den Vorteil, von einfachem Verständnis und der schieren Menge an Material, zunichte.

Sollte es dabei dann zu Problemen kommen, gibt es keinen direkten Ansprechpartner, der mit Rat und Tat zu Seite steht und dabei die Monopolstellung innehat.

Im Bezug zur Sicherheitsfrage gibt es hier natürlich auch Bedenken, dass durch die Veröffentlichung von Schnittstellen und Programmen, Angriffe deutlich leichter durchzuführen sind. [8]

5.4 Beispiele

Als wichtige Beispiele sind hier die Betriebssysteme Linux [1] und Android anzuführen, welche beide einen riesigen Einfluss auf das jeweilige Zielgebiet haben. Aber auch andere Programme wie OpenOffice, VLC Mediaplayer oder der Firefox Browser sind Paradebeispiele für weltweit integrierte Programme aus OpenSource-Quellen.

6 Beispiel BIBIFI – Build IT, Break It, Fix It

6.1 Idee/Übersicht

Die Idee hinter dem BIBIFI – dem Build It, Break It, Fix It – Wettbewerb ist es, im Gegenteil zu anderen Wettbewerben, ein Programm nicht nur zu knacken, sondern auch vorher zu schreiben. Verschiedene Teams arbeiten in verschiedenen Phasen gegeneinander um jeweils einen höheren Punktestand zu erreichen. Dadurch sollen nicht nur die zerstörerischen, sondern auch die schaffenden Fähigkeiten der Teilnehmer getestet werden. Außerdem wird dadurch auch die Wichtigkeit von gutem Programmierstil und Implementierung, sowie auch Programmiersprache aufgezeigt. [5]

Ausgetragen wurden drei verschiedene Wettbewerbe im Jahr 2015, welche in den nächsten Abschnitten behandelt werden.

6.2 Ziel

Das Ziel dieses Wettbewerbes war es, herauszufinden, ob gute Softwareknacker auch gleichzeitig gute Softwareentwickler sein können. Außerdem soll der Bau von sicherer Software, den weltweit agierenden Teams, näher gebracht werden, sodass diese in ihren zukünftigen Projekten vermehrt ein Auge auf mögliche Sicherheitsrisiken haben, welche sie möglicherweise vorher nicht hatten.

Der wissenschaftliche Aspekt hiervon liegt aber klar auf der Evaluierung der unterschiedlichen Kenntnisse der Teilnehmer und die Auswirkung davon auf die Software.

6.3 Umfeld

6.3.1 Wer nimmt teil?

Contest	Spring '15†	Fall '15†	Fall '15
# Contestants	156	122	23
% Male	91%	89%	100%
% Female	5%	9%	0%
Age	34.8/20/61	33.5/19/69	25.1/17/31
% with CS degrees	35%	38%	23%
Years programming	9.6/0/30	9.9/0/37	6.6/2/13
# Build-it teams	61	34	6
Build-it team size	2.2/1/5	3.1/1/5	3.1/1/6
# Break-it teams	65	39	4
(that also built)	(58)	(32)	(3)
Break-it team size	2.4/1/5	3.0/1/5	3.5/1/6
# PLs known per team	6.8/1/22	10.0/2/20	4.2/1/8

Abbildung 1: Statistische Auffassung aller Teilnehmer [5]

Der Wettbewerb wurde von Professoren der University of Maryland in Zusammenarbeit mit Professoren der Carnegie Mellon University entwickelt. Die Professoren kamen vorrangig aus der IT und Sicherheitsbranche. Teilnehmer des Wettbewerbs sind Menschen mit einem großen Interesse an IT-Sicherheit und Programmieren, vorrangig von Universitäten oder aus Firmen. Jedoch wurde der Wettbewerb auch in Zusammenarbeit mit einem MOOC ausgetragen. Durch den MOOC, einem Massive Open Online Course, wurde die Teilnehmerzahl deutlich erhöht, wie in Abbildung 1 anhand von Spring '15 und Fall '15(1) zu sehen ist.

6.3.2 Was wird gemacht?

Der Wettbewerb ist in 3 Phasen mit jeweils 2 Wochen Zeit eingeteilt. Anfänglich wird eine Aufgabe mit Sicherheitsrelevanz gestellt, die beinhaltet, ein definiertes Problem programmiert umzusetzen. Diese Aufgabe muss in der ersten Phase, der Build It Phase, von den Teams erfüllt werden. Während dieser Zeit laufen Tests über die Abgaben und erzeugten Punkte anhand vom Komitee definierten Regeln. Nach dem Ablauf der 2 Wochen werden alle Einsendungen vermischt und von den Break It Teams in den anschließenden 2 Wochen in der Break It Phase unter die Lupe genommen und Sicherheitsmängel identifiziert. Auch hier werden anhand von Relevanz des Mangels Punkte vergeben. Diese Mängel werden protokolliert und an die Build It Teams weitergeleitet, sodass diese in der letzten Phase so viele Fehler wie möglich beheben können und somit ihren Punktestand auf ein finales Level bringen können.

6.3.3 Wie wird es gemacht?

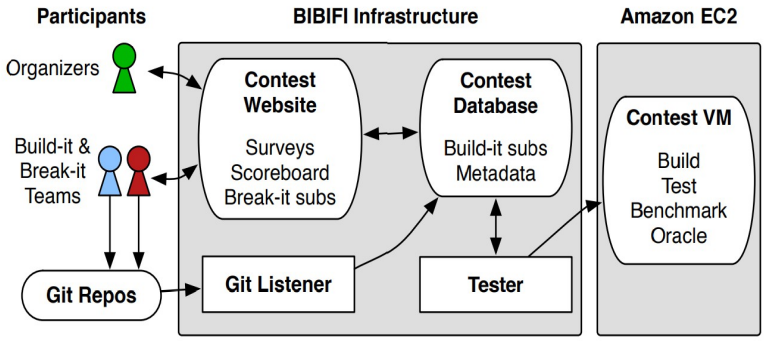

Abbildung 2: Infrastruktur des BIBIFI [5]

Der BIBIFI hat in der Entwicklung einen großen Aspekt darauf gelegt, den Aufwand der Betreiber so gering wie möglich zu halten. Daher gibt es während des Wettbewerbs keine persönlichen Treffen oder Vorträge. Dies hat den Vorteil, dass dadurch Interessierte aus der ganzen Welt teilnehmen können, wie in Abbildung 3 zu sehen.

Contest	USA	Brazil	Russia	India	Other
Spring 2015	30	12	12	7	120
Fall 2015	64	20	12	14	110

Abbildung 3: Herkunft der Teilnehmer [5]

Die Infrastruktur des BIBIFI, wie Abbildung 2 zeigt, ist in zwei große Teile aufgeteilt. Der „BIBIFI Infrastructure“ - Teil ist ein Webinterface dass auf den Servern der University of Maryland liegt. Hier ist ein großer Aufwand betrieben worden, um ein sicheres Web-Frontend zu schaffen, welches zum einen gegen böswillige Teilnehmer oder Außenstehende davon abhält den Wettbewerb zu stören, zum anderen aber auch den Teilnehmern genug Möglichkeiten gibt, Tests einzureichen, Punktestände einzusehen und Umfragen auszufüllen.
Der zweite Teil ist ein in der Amazon EC2 Cloud liegendes Linux Ubuntu System auf welchem alle Abgaben laufen müssen und welches die Tests durchführt. Dieser Teil wird auch Oracle genannt, da es die Abgabe evaluiert und ein Urteil durch Punktevergabe fällt. Die Tests des Oracles starten automatisch, nachdem eine teilnehmende Gruppe einen Push in ihr eigenes Git-Repository durchführt und dadurch eine neue Version ihres Programms herausgibt.

6.4 Erste Phase

In der ersten Phase, der **Build It Phase**, muss die Aufgabenstellung umgesetzt werden. Die Aufgabenstellung ist daran angepasst, dass ein durchschnittlicher Entwickler so eine Aufgabe in zwei Wochen gut und sinnvoll implementieren kann und trotzdem gut zu testen und evaluieren ist. Die Implementierung der Aufgabe hat im Idealfall alle Extras an Funktionalität, keine Sicherheitslücken und ist in Sachen Performance einwandfrei. Die drei Punkte sind ausschlaggebend für einen guten Punktestand, denn nach jedem Push den eine Gruppe in ihr Git-Repository macht, läuft ein Test automatisch ab, der anhand der bestandenen Tests Punkte vergibt und diese in ein Scoreboard einträgt.

6.5 Zweite Phase

In der zweiten Phase, der **Break It Phase**, werden alle Abgaben der ersten Phase vermischt und den Break It Teams zur Verfügung gestellt. In dieser Phase können mehr Teams teilnehmen, da nicht jedes Team das in dieser Phase teilnimmt auch gebaut haben muss. In den hierfür verfügbaren zwei Wochen sollen die Teams versuchen in den Abgaben Sicherheitsprobleme und andere Schwachstellen zu finden und diese dann als Fehlerbericht einzusenden. Diese Fehlerberichte werden dann vom Oracle gegen gecheckt und entweder akzeptiert oder verworfen.

Die Abgaben müssen vorher alle Tests des Oracles bestanden haben, müssen also auch funktionieren und dürfen nichts verschleiern. Verschleierung hier bedeutet, dass der Quellcode für Menschen lesbar und verständlich, sowie eine klarer Programmablauf zu erkennen sein muss. Sollte dies nicht erfüllt sein, dürfen die Abgaben der Teams nicht weiter teilnehmen.

Jedes Team in der Break It Phase darf auf alle Abgaben zugreifen, sodass es keine Ungleichheit gegenüber anderen Abgaben entsteht. Jedoch ist es nicht möglich sich nur auf eine Abgabe einzustellen, denn die Anzahl an möglichen Einsendungen pro Software sind begrenzt.

Punkte werden in dieser Phase anhand der Einzigartigkeit und der Relevanz des Fehlers gemessen. Ist der Fehler schwerwiegender oder einzigartig, gibt er mehr Punkte und durch die Begrenzung der Einsendungen, liegt der Fokus hier klar auf solchen Fehlern.

6.6 Dritte Phase

Die letzte Phase, die **Fix It Phase**, erlaubt es den Teilnehmern die eine Einsendungen gebaut haben, anhand der, in der Break It Phase entstandenen, Fehlerberichte ihre Software zu verbessern und sicherer zu machen. Außerdem können sie hier die Punkte, die ihnen durch die Break It Phase abgezogen wurden, wieder ausgleichen. Die Teams dürfen mit jedem Fix nur einen bestimmten Fehler beheben, andernfalls werden die Fehler, die behoben wurden, als gleich angesehen und deshalb nur ein Fehlerpunkt abgezogen. Die Software muss aber weiterhin alle Kriterien aus der Build It Phase einhalten.

6.7 Punkteberechnung

6.7.1 Build It Scores

Die Punkteberechnung in der Build It Phase wird vom Oracle übernommen. Die zu erreichenden Punkte werden in zwei Bereiche eingeteilt: die Sicherheitspunkte (resilience score) und die Effizienz und Korrektheitspunkte (ship score). Die Effizienz- und Korrektheitspunkte werden direkt nach der Build It Runde berechnet. Das Oracle lässt vorgegebene Test durchlaufen und anhand der Testergebnisse werden Punkte verteilt. Korrektheitspunkte geben standardmäßig *M* Punkte, optionale Korrektheitspunkte geben *M/2* Punkte, wobei *M* hier eine festgelegte Zahl ist, die dem Test entsprechend ist. Effizienzpunkte geben entsprechend der gemessenen Performance ihre Punkte. *M · (Schlechtester - Wert)/(Schlechtester - Bester)* gibt hier einen Wert an, wobei *M* weiterhin die entsprechende Testpunktzahl ist und der *Wert* die gemessene Zeit ist. *Schlechtester* und *Bester* sind die Werte der am besten und am schlechtesten performenden Abgabe. Daher liegen die Punkte hier zwischen *0* und *M*.

Die Sicherheitspunkte werden erst nach der Fix It Phase definitiv berechnet, da erst dann alle noch vorhandenen Sicherheitsmängel bekannt sind. Da die Sicherheitspunkte negativ gerechnet werden, ist hier der beste Wert *0.* Sind am Ende des Wettbewerbs noch Sicherheitsprobleme vorhanden, gibt es für Korrektheitspunkte *M/2* Punkte, für Speichersicherheitsprobleme *M* Punkte und für Exploits oder andere Sicherheitsprobleme *2M* Punkte Abzug.

6.7.2 Break It Scores

In der Break It Phase kann jedes Team Punkte sammeln, indem sie Fehler in anderen Programmen finden. Das Ziel ist, so viele Fehler wie möglich zu finden, um am Ende auch so viele Punkte wie möglich zu bekommen. Damit aber die Teams nicht nur offensichtliche Fehler einreichen, werden Sicherheitsfehler gegenüber Korrektheitsfehlern mit vier zu eins gerechnet und wenn Fehler von mehreren Gruppen eingereicht werden, werden die Punkte pro Fehler durch die Anzahl an Gruppen geteilt. Pro gefundenem Fehler bekommen die Gruppen *P* Punkte zugeteilt anhand der in Kapitel 6*.7.1* beschriebenen Einteilung. Diese Maßnahmen sollen die Gruppen dazu auffordern mehr nach den „Hard-To-Find“ Fehlern zu suchen anstatt nur die „low-hanging-fruits“ zu suchen.

6.7.3 Abrechnung

Um den endgültigen Wert eines Teams zu berechnen, werden am Ende alle Punkte zusammengerechnet. Dafür eignet sich folgende Formel:

Endergebnis = Build It Score - Break It Score (d.g.G.) - Fix It Score (d.e.G.))

+ (Break It Score (d.e.G.) - Fix It Score (d.g.G))

d.g.G.: Der gegnerischen Gruppe, also Punktzahl der Gruppe die die eigene Software geknackt oder deren eigene Software verbessert hat.

d.e.G.: Der eigenen Gruppe, also die Punktzahl durchs Knacken der gegnerischen Software oder durchs verbessern der eigenen Software.

6.8 Auswertung

6.8.1 Programmiersprachenabhägigkeit

Interessant bei der Auswertung des Wettbewerbs ist die Tatsache, wie unterschiedlich die verschiedenen Programmiersprachen abgeschnitten haben. Sprachen wie *C/C++* schlossen in Hinsicht auf Korrektheit und Performance deutlich besser ab, jedoch hatten sie auch als einzige Sprache Fehler im Speichermanagement. Wie Abbildung 4 zeigt, schnitten dynamische Programmiersprachen wie *Python* oder *PHP* in Integritäts- Fehlern deutlich schlechter ab als statische Sprachen wie *Java* oder *C#.*

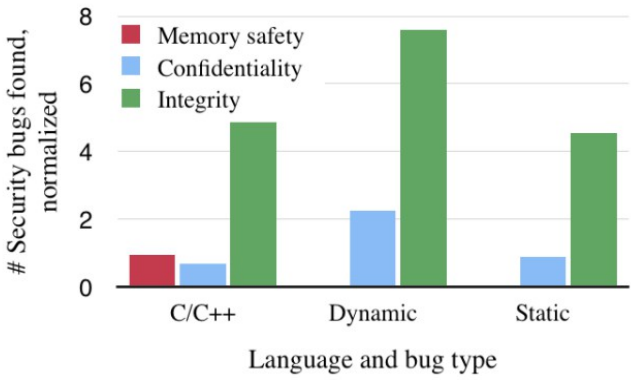

Abbildung 4: Programmiersprache in Abhängigkeit zur Fehlerart [5]

Die dynamischen Sprachen waren die kürzesten Lösungen, jedoch wurde festgestellt, dass mehr Zeilen Code nur geringfügig zu mehr Sicherheitslücken und Fehlern geführt haben.

6.*8.2 Erkenntnisse*

Die drei Wettbewerbe haben einige Dinge klar aufgezeigt. An erster Stelle ist und bleibt, dass jeder Fehler macht. Selbst wenn die Aufgaben mit einem Augenmerk auf Sicherheit zu lösen waren, sind in jeder Einsendung Fehler zu finden gewesen. Auch gab es Gruppen die zwar das Programm richtig implementiert haben, jedoch in Sachen Sicherheit wichtige Dinge vergessen haben, wie zum Beispiel den Schlüssel zum Verschlüsseln einer Kommunikation, statisch zu behalten. Im Gegensatz dazu ist aber auch aufgefallen, dass je mehr Sprachen in einem Team vorhanden waren und je mehr Gruppenmitglieder dabei waren umso besser schnitt das Team ab. Diese Erkenntnis lässt sich darauf zurückführen, dass jede Sprache Eigenschaften hat, die gut und schlecht sind und daher die Programmierer an mehr mögliche Fehler denken.

6*.8.3 Probleme*

Da dieser Wettbewerb noch in den Kinderschuhen steckt, sind Probleme nicht zu vermeiden. Aufwand für die Veranstalter, Limitation von Einsendungen, Preisvergabe und Punktevergabe wurden nach dem ersten Durchlauf angepasst, um ein besseres Ergebnis zu erzielen. Diese Änderungen haben dazu geführt, dass deutlich mehr Sicherheitsfehler gefunden wurden, und dadurch sogar die Programme sicherer wurden.

6.9 Letzte Events

6.9.1 2015/1

Der im Frühjahr 2015 ausgeführte Wettbewerb hatte als Hauptsicherheitsthema, Dateisicherheit und sichere Prozesskommunikation. Es ging darum, dass die Teilnehmer zwei Programme schreiben sollten, welche eine Kunstgalerie überwachen sollen. Das eine Programm sollte ein Log schreiben, das andere das Log lesen können.

6.9.2 2015/2/3

Im Herbst 2015 fanden zwei weitere Wettbewerbe statt. Der erste war offen für MOOC Teilnehmer und der zweite nur für US-Studenten und US-Absolventen. Für beide war die Aufgabe, ein System zu bauen, welches einer Bank ähnelt und ein anderes, welches einem Bankautomaten ähnelt. Die Schwierigkeit hierbei war es, die Kommunikation von beiden, welche über ein Netzwerk lief, richtig abzusichern und zu schützen. Außerdem wurde hier ein erhöhtes Augenmerk auf Performance gelegt, damit die Kommunikation nicht zu lange dauerte.

7 Fazit

7.1 Fragestellung im Bezug zum BIBIFI

Der BIBIFI hat bewiesen, dass selbst erfahrene Entwickler in einem Umfeld wo Sicherheit eine große Rolle spielt, kein Programm erzeugen können, welches zu 100% sicher ist. Menschen machen Fehler und das wird auch weiterhin so bleiben. Dieser Wettbewerb ist ein Paradebeispiel dafür, wie ein zukünftiges Modell der sicheren Entwicklung aussehen könnte. Der Wettbewerb bewirkt anfänglich das Konkurrenzverhalten zwischen den Gruppen ein sicheres System zu bauen, dass am besten alles kann und so sicher wie möglich ist. Zum Ende hin findet eine öffentliche Evaluierung des geschriebenen Codes statt, um diesen gründlich zu testen und zu evaluieren. Dabei werden dann deutlich mehr Sicherheitsprobleme gefunden und so die Software deutlich sicherer gemacht.

7.2 Zukunftsidee

Als Zukunftsidee wäre es möglich, Programme weiterhin im Geheimen zu entwickeln, um dadurch einen Wettbewerb zwischen den Firmen zu erhalten und um Geheimnisse zu schützen. Es sollte möglich sein, eine Art öffentlichen Test zu generieren. Möglich wäre, dass jede Firma einen Wettbewerb, wie den BIBIFI, ausrichtet und mit Preisgeldern Leute motiviert ihre Software zu testen. Dadurch wäre es möglich, eine vertrauenswürdige Software zu erzeugen, die auch sicher ist. Es könnte wie eine Art Beta-Test funktionieren nur statt für Spiele oder Oberflächenänderungen, für Sicherheitssoftware und Systeme: ein Härtetest für das darunterliegende System. Einige Unternehmen haben diese Idee bereits umgesetzt, aber große Teile der Gesellschaft wissen davon nichts.
Möglich wäre aber auch, dass gewisse Teile eines Programms gleich sein müssen. Wie zum Beispiel das OSI-Modell. Jede Internet-Applikation nutzt es und es ist ein Standard. Würde es so etwas Einheitliches für einen Messenger oder eine andere Sicherheitsapplikation geben, dann würde ein deutlich größerer Teil offengelegt werden. Die Unternehmen könnten sich weiterhin durch Features und Oberfläche von einander abheben.
Daher möchte ich meine eingehende Fragestellung - ob eine öffentliche Sicherheitsentwicklung mehr Sinn macht als geheime - damit beantworten, dass eine Mischung aus Beidem am wahrscheinlichsten aber auch am sinnvollsten ist.

Quellen:

[1] The Linux Foundation(2017):Linux Kernel Development Report

https://go.pardot.com/l/6342/2017-10-24/3xr3f2/6342/188781/Publication_LinuxKernelReport_2017.pdf (19.09.2018; 10:11)

[2] Domenico Raguseo(10.02.2017):The Future of Cybersecurity

https://securityintelligence.com/the-future-of-cybersecurity/ (01.06.2018; 11:20)

[3] SentinelOne(10.03.2017): THE HISTORY OF CYBER SECURITY – EVERYTHING YOU EVER WANTED TO KNOW

https://www.sentinelone.com/blog/history-of-cyber-security/ (01.06.2018; 11:21)

[4] Stefan Krempl(13.09.2017): Kampagne Public Code: Software für die Verwaltung soll frei sein
https://www.heise.de/newsticker/meldung/Kampagne-Public-Code-Software-fuer-die-Verwaltung-soll-frei-sein-3830705.html (01.06.2018; 11:24, Stefan Krempl)

[5] Ruef A. , Hicks M. , Parker J., Levin D. , Mazurek M., Mardziel P. (2016): Build It, Break It, Fix It: Contesting Secure Development
https://arxiv.org/pdf/1606.01881v2.pdf (01.06.2018; 11.25)

[6]Saraswati Experts (2016):COMPUTER SCIENCE WITH C++
https://books.google.de/books?id=OtlBDAAAQBAJ&pg=SA1-PA31&redir_esc=y#v=onepage&q&f=false (18.09.2018; 13:36)

[7] BIBIFI: Contest Details
https://builditbreakit.org/details (01.06.2018;11:27)

[8] Susanne Moog(13.07.2017): Pros & cons of open-source software at the enterprise level
https://typo3.com/blog/pros-cons-of-open-source-software-at-the-enterprise-level/ (01.06.2018;11:29)

[9] Michael Nadeau (19.09.2017): Future cyber security threats and challenges: Are you ready for what's coming?
https://www.csoonline.com/article/3226392/security/future-cyber-security-threats-and-challenges-are-you-ready-for-whats-coming.html (01.06.2018;11:30)

[10] Bisk: The History of Information Security
https://www.villanovau.com/resources/iss/history-of-information-security/
(01.06.2018;11:30)

[11] Matthias Springer(08.09.2016): Was ist der Unterschied zwischen Safety und Security?
https://www.tuev-nord.de/explore/de/erklaert/was-ist-der-unterschied-zwischen-safety-und-security/ (01.06.2018;11:33)

[12] National Cyber Security Center: Secure development and deployment
https://www.ncsc.gov.uk/guidance/secure-development-and-deployment
(01.06.2018;11:34)

[13] Michael Roesner (19.06.2015): Virenlexikon: Creeper – Catch me if you can!
https://www.kaspersky.de/blog/virenlexikon-creeper-catch-me-if-you-can/5368/
(02.06.2018;13:08)

[14] Stefanie Schäfers (12.07.2018): Cloud-Sicherheit im Fokus
https://it-service.network/blog/2018/07/12/deutsche-cloud-sicherheit/ (18.09.2018;
17:33)

[15] Redaktion InterFace AG (19.02.2018): WARUM DIE ANGST VOR DER DSGVO
ÜBERTRIEBEN IST
https://www.interface-ag.com/digitale-transformation/warum-die-angst-vor-der-dsgvo-uebertrieben-ist/ (18.09.2018; 17:36)

[16] Tomas Rudl (26.01.2018) : Staatstrojaner: Das große Schnüffeln hat begonnen
https://netzpolitik.org/2018/staatstrojaner-das-grosse-schnueffeln-hat-begonnen/
(18.09.2018; 17:38)

[17] Daniel Berger(15.05.2018): Facebook: Neues Datenleck betrifft 3 Millionen Nutzer
https://www.heise.de/newsticker/meldung/Facebook-Neues-Datenleck-betrifft-3-Millionen-Nutzer-4049832.html (18.09.2018; 17:40)

[18] Dirk Liedtke(26.10.2016): Wie fremde Kontakte aufs Iphone gelangten
https://www.stern.de/digital/smartphones/apple-datenleck--wie-fremde-kontakte-aufs-iphone-gelangten-7118518.html (18.09.2018; 17:42)

[19] Fraunhofer SIT : Volksverschlüsselung
https://volksverschluesselung.de/ (18.9.2018; 17:52)

[20] BSI / BMI: De-Mail
https://www.de-mail.info/ (18.09.2018; 17:52)

[21] Microsoft Corporation(2005): Entwicklungszyklus für sichere Software
https://msdn.microsoft.com/de-de/en-en/library/ms995349.aspx (18.09.2018; 18:32)

[22] Heiko Lossie (23. 06.2007): Wie 75 Cent zum Verhängnis wurden
https://www.stern.de/digital/computer/20-jahre--kgb-hack--wie-75-cent-zum-verhaengnis-wurden-3269216.html (18.09.2018; 18:33)

[23] GNU.org: Proprietäre Software ist häufig Schadsoftware
http://www.gnu.org/proprietary/proprietary.de.html (19.09.2018; 12:43)

[24] BlackDuck:Open Source Vulnerability Management
https://www.blackducksoftware.com/open-source-vulnerability-management (19.09.2018;
12:52)

[25] Martin Holland (29.04.2014): Heartbleed: US-Regierung hält IT-Schwachstellen geheim
https://www.heise.de/security/meldung/Heartbleed-US-Regierung-haelt-IT-Schwachstellen-geheim-2179033.html (19.09.2018; 13:12)

[26]Sam Saltis (02.08.2018)
https://www.coredna.com/blogs/comparing-open-closed-source-software
(19.09.2018;13:22)

[27] Khrystyna Oliinyk (03.05.2018):5 Differences Between Open Source and Closed Source Software
https://api2cart.com/business/5-differences-between-open-source-and-closed-source-software/ (19.09.2018; 13:45)

[28] Chris Burnett : OPEN SOURCE VS CLOSED SOURCE – WHICH IS MORE SECURE?
http://www.franklinfitch.com/blog/2017/06/13/open-source-vs-closed-source-secure/
(19.09.2018; 14:20)

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum Unterschrift (Vor- und Nachname)