esse

ESSE Einführung in Security – VO 08: Testing

Thomas Stipsits, Florian Fankhauser, Christian Brem

24W

ESSE (Establishing Security) – IT Security Research Team
Research Group for Industrial Software (INSO)
https://establishing-security.at/

Agenda

- Grundlagen des Testens
- Merkmalsräume von Software-Tests
  - Prüfkriterium
  - Prüfebene
  - Prüfmethodik
- Testtechniken
  - Statische Codeanalyse
  - Fuzzing
  - Penetration Testing

Testabdeckung vs. Produktqualität

(Vergleiche `http://turnoff.us/geek/the-depressed-developer-17/`)

Grundlagen des Testens

Einführung – Testing Definitionen

„Testing is the process of executing a program with the intent of finding errors“ (Myers und Sandler (2004))

„Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence“ (Dijkstra (1972))

„In essence, a web application that fails any of the tests put to it by an application scanning tool is truly one in bad shape. On the other hand, a web application that passes all of the tests still can’t be declared secure in any significant sense.“ (van Wyk (2007))

Testen – Studie Carnegie-Mellon-Universität

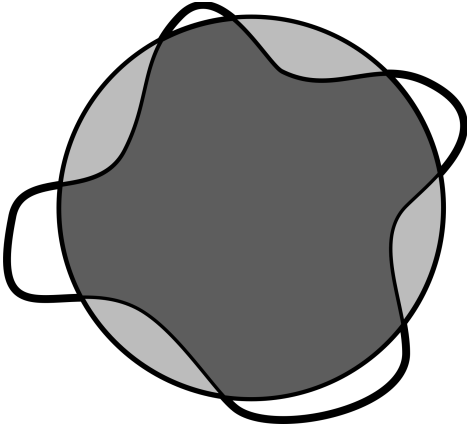
- „Tausend Zeilen Programm ca. fünf bis fünfzehn Bugs – nach dem Testen“
  - Diese bleiben oft unentdeckt
  - Haben keine Auswirkung auf die restliche Funktionalität
  - Stellen mögliche Sicherheitsfehler dar
  - Gezielte Suche nach diesen Fehlern erforderlich
- Test ist eine Momentaufnahme und zeitlich beschränkt → Angreifer hat beliebig Zeit, dynamische Entwicklungen
- Testen alleine nicht ausreichend für ein sicheres System
- Testen ist ein Teil zur Erstellung eines sicheren Systems
- Oftmals durch bestimmt Faktoren getrieben → Kosten/Nutzen

Was ist Testen?

- Überprüfen des aktuellen Zustands mit einem geplanten Zustand
- **Validierung**: Erzeugen wir das richtige Produkt?
  - Vergleich: Wünsche des Kunden $\longleftrightarrow$ Anforderungen
- **Verifikation**: Erzeugen wir das Produkt richtig?
  - Vergleich: Anforderungen $\longleftrightarrow$ Applikation
- Zielsetzung des Tests ist die Auffindung von Fehlern
- Unterscheidung nach Testziel
  - Funktionalität, Sicherheit, Usability, ...

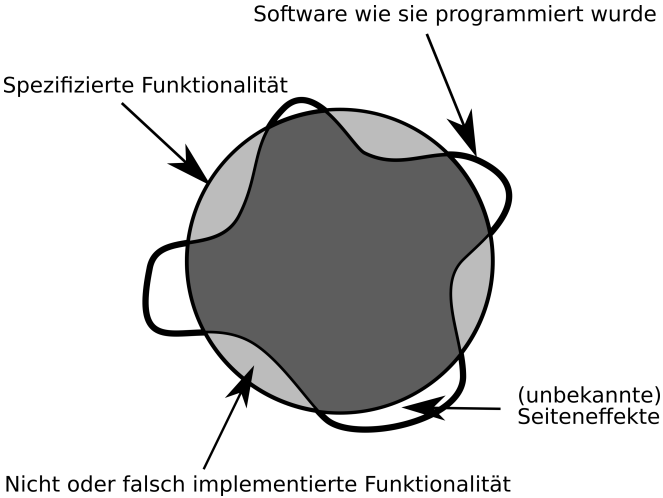
Software Fehler

- Anforderungen
- Implementierung
- Seiteneffekte
- Fehlende/falsche Implemenentierung



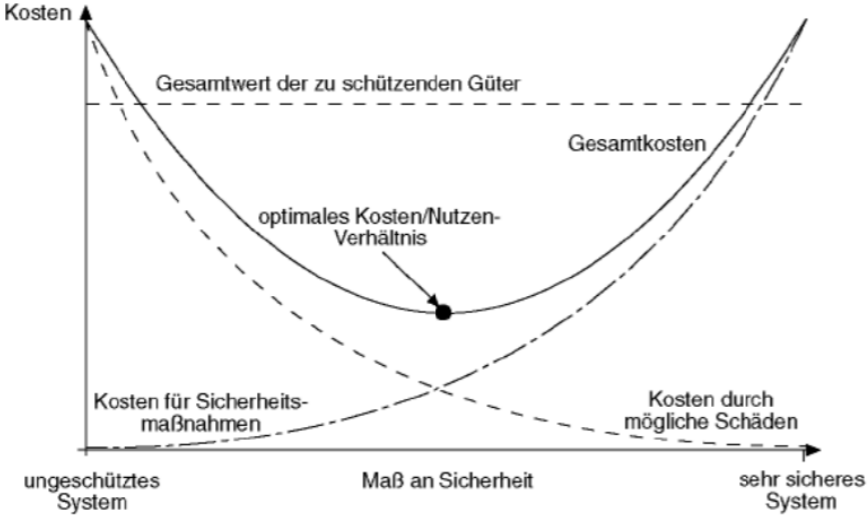
(Vergleiche Thompson (2003))

Software Fehler – Auflösung



(Vergleiche Thompson (2003))

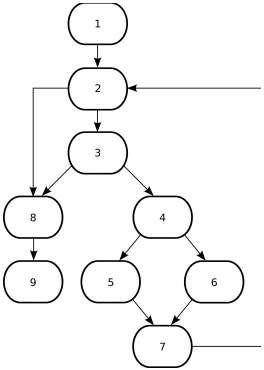
Kosten-/Nutzenanalyse



(Vergleiche M. Raepple: Sicherheitskonzepte fur das Internet)

Testabdeckung

- Wieviele Teile eines Systems wurden getestet
- Berechnung nach Zeilen, Anweisungen, Datenvarianten, usw.
- Erreichung von 100% dabei nahezu unmöglich



Metriken

(Vergleiche `http://www.osnews.com/story/19266/WTFs_m`)

Wer kann Sicherheitstests durchführen?

- **Entwickler:in/Tester:in**
  - Fehlendes Wissen über Sicherheit und aktuelle Schwachstellen → Schwachstellen werden leicht übersehen
  - Augenmerk auf Funktionalität
  - Oft Implikationen aufgrund von detaillierten Wissen über Code
  - Testen mit Implikationen kann zu Fehlannahmen führen führen
- **Sicherheitsexpert:in**
  - Fehlendes Wissen über Domäne und die Implementierungsdetails → komplexe Logikflüsse werden leicht übersehen
  - Wissen über Schwachstellen und aktuelle Entwicklungen im Sicherheitsbereich wie zum Beispiel Sicherheits-Testtools
  - Testen des Systems aus Angreifersicht → Realitätsnäher

Merkmalsräume von Software-Tests

Merkmalsräume von Software-Tests

- Charakterisierung unterschiedlicher Testarten
- Betrachtung der Menge aller möglichen Tests aus unterschiedlichen Blickwinkeln:
  - **Prüfkriterium**
    - Inhaltliche Orientierung des Testfalls (**WAS?**-Frage)
  - **Prüfebene**
    - Repräsentiert derzeitige Phase im Lebenszyklus der Software (**WANN?**-Frage)
  - **Prüfmethodik**
    - Welche Methoden und Techniken sollen angewandt werden? (**WIE?**-Frage)

(Vergleiche Hoffmann (2008))

Merkmalsraum Prüfkriterium

- ***Prüfkriterium***
  - Inhaltliche Orientierung des Testfalls
  - **Funktionale Tests:**
    - Testen Funktionen des Systems
    - z.B Login, PKI, Speicherung von Daten, Sicherheitsfunktionen, ...
  - **Nicht-funktionale Tests:**
    - Eigenschaften eines Systems
    - z.B Zuverlässigkeit, Robustheit, Sicherheit, ...
- ***Prüfebene***
  - Auf welcher Ebene soll geprüft werden?
  - Repräsentiert derzeitige Phase im Lebenszyklus der Software
- ***Prüfmethodik***
  - Welche Methoden und Techniken sollen angewandt werden?

(Vergleiche Hoffmann (2008))

Funktionale vs. nicht-funktionale (Sicherheits-)Tests

- **Funktionale (Sicherheits-)Tests**
  - Spezifikation als Grundlage → Erwarteter Output
  - Tests sind in sich abgeschlossen
  - Funktionale Fehler können Auswirkung auf Sicherheit haben
  - Testen der Bereitstellung der korrekten Funktionalität
  - *z.B. Login mit gültigen Credentials möglich?*
- **Nicht-funktionale (Sicherheits-)Tests**
  - Spezifikation nicht nutzbar als Grundlage
  - Abuse Cases → Robustheit gegen Angriffe
  - Tests beeinflussen sich gegenseitig
  - Sicherheitsfehler können in jeder Funktionalität auftreten
  - Testen der Robustheit gegen Angriffe
  - *z.B. Login mit ungültigen Credentials möglich? (z.B. SQLI)*

Merkmalsraum Prüfebene

- ***Prüfkriterium***
  - Inhaltliche Orientierung des Testfalls
  - Funktionale Tests
  - Nicht-Funktionale Tests
- **Prüfebene**
  - Auf welcher Ebene soll geprüft werden?
  - Repräsentiert jeweilige Phase im Lebenszyklus der Software
- ***Prüfmethodik***
  - Welche Methoden und Techniken sollen angewandt werden?

(Vergleiche Hoffmann (2008))

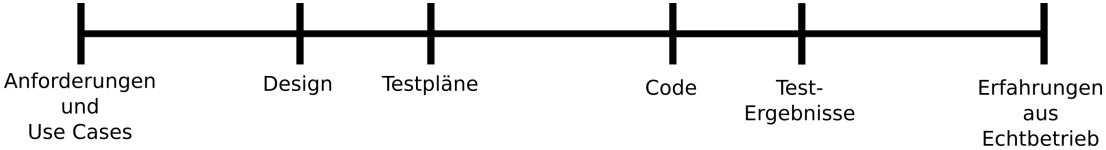
Prüfebene Eigenschaften

- Jede Phase des Lebenszyklus besitzt eigene Eigenschaften, Deliverables und Herausforderungen
- Unterschiedliche Testarten finden in unterschiedlichen Phasen statt & finden unterschiedliche Fehler
- **Fehler in frühen Lebensphasen** können in späteren Phasen
  - gravierende Auswirkungen haben
  - Folgefehler verursachen
  - schwer zu entdecken & beheben sein

  Folgefehler verursache
- Daher: Testdurchführung: Je früher desto Besser!
- Um flächendeckende Tests durchzuführen müssen Tests gut geplant werden
- **Zielsetzung** von Tests beachten!

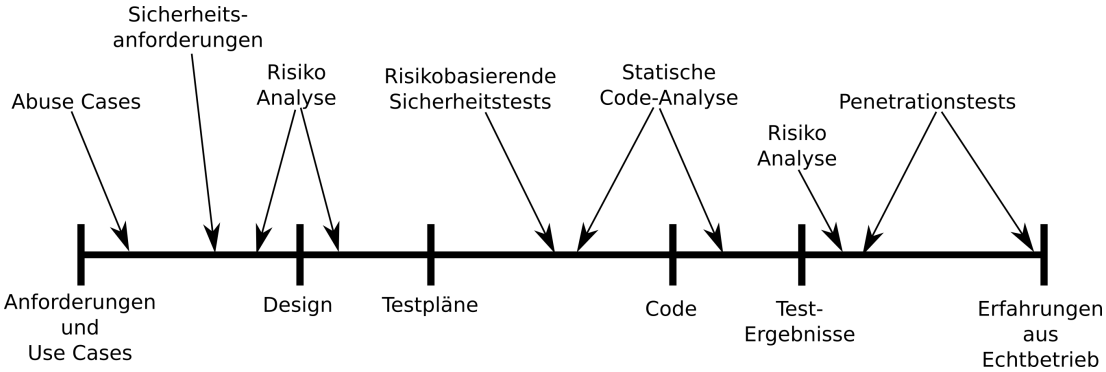
Sicherheitstests im Lebenszyklus

- Statische-Code-Analyse
- Penetrationstests
- Abuse Cases
- Risiko Analyse



(Vergleiche Chess und McGraw (2004))

Sicherheitstests im Lebenszyklus – Auflösung



(Vergleiche Chess und McGraw (2004))

Merkmalsraum Prüfmethodik

- ***Prüfkriterium***
  - Inhaltliche Orientierung des Testfalls
  - Funktionale Tests
  - Nicht-Funktionale Tests
- ***Prüfebene***
  - Auf welcher Ebene soll geprüft werden?
  - Repräsentiert derzeitige Phase im Lebenszyklus der Software
- ***Prüfmethodik***
  - Welche Methoden und Techniken sollen angewandt werden?
  - Welches Wissen steht dem Testenden zur Verfügung?
  - Was sind die Annahmen für die Testdurchführung?

(Vergleiche Hoffmann (2008))

White-Box-Tests

- **Wissen über Systemdetails** wird für Tests herangezogen
  - Verwenden von Source Code, Dokumentationen, Konfigurationsdetails, ...
  - Wissen über „Erwartetes Verhalten“ vorhanden
  - Ableiten von Ein- und Ausgabewerten
  - Prüfen von tatsächlichen Verhalten $\longleftrightarrow$ erwarteten verhalten
- Effizient in der Auffindung von Fehlern und Abweichungen, die durch statische Tätigkeiten aufgefundne werden können
- Ineffizient in der Auffindung von Logikfehlern, die nur durch dynamische Testtechniken aufgefunden werden
- Zugriff auf Informationen nicht immer möglich $\rightarrow$ vollständige Informationen selten vorhanden
- z.B. Statische Analyse, Code Reviews, ...

(Vergleiche Dustin et al. (2001))

Black-Box-Tests

- **Wissen über Systemdetails** wird für Tests **NICHT** herangezogen
  - Wissen über „Erwartetes Verhalten“ muss erarbeitet werden
  - Definition und Beobachtung von Ein- und Ausgabewerten
  - Prüfen des tatsächlichen Verhaltens → Deduktion ob Verhalten problematisch ist
- Vergleich der tatsächlichen mit spezifizierter/erwarteter Ausgabe
- Eher ineffiziente Durchführung - dafür aber realitätsnahe
- Erforderlich wenn kein Zugriff auf Code vorhanden ist
- Alle Datenvarianten nicht möglich
  - Äquivalenzklassen, beispielsweise:
    - Verwendung eines Vertreters aus Menge der positiven und negativen Integer
  - Grenzwertanalyse, beispielsweise:
    - Vertreter aus Grenzbereichen (MAX Integer und MIN Integer)

(Vergleiche Dustin et al. (2001))

Gray-Box-Tests

- Kombination von White-Box- und Black-Box-Testing
- Häufig präferiertes Verfahren für die Testdurchführung
  - Durch Informationen können Tests optimiert werden
  - Ermittlung der Testabdeckung erleichtert
- Datenvarianten anhand interner Details identifizieren (White-Box-Tests)
- Tests mit ermittelten Datenvarianten durchführen (Black-Box-Tests)

(Vergleiche Dustin et al. (2001))

Testtechniken

Statische Codeanalyse

- Untersuchung des Codes ohne Ausführung
  - Lauffähigkeit des Systems nicht erforderlich
  - Keine spezielle Testumgebung
- Manuelle Analysemethoden
  - Secure Code Review
  - Problem ist der Aufwand
- Automatisierte Analysemethoden
  - Compiler, automatisierte Scanner, ...
  - Problem ist hohe false-positiv Rate, false-negatives etc.

Secure Code Review

- Durchführung kostenintensiv aber potentiell sehr effektiv
- **Top-Down Ansatz**
  - Ausgangspunkt: Wissen über Schwachstellen
  - Erforderliches Detailwissen über Code gering
- **Bottom-Up Ansatz**
  - Ausgangspunkt: Wissen über Code
  - Höherer Aufwand als beim Top-Down Ansatz
- Walkthrough: Testfälle am Papier durchführen
- Inspection: Überprüfung anhand von Checklisten/Coding Standards
- Einfaches Design/Implementierung vereinfacht ein Review

(Vergleiche Chess und McGraw (2004))

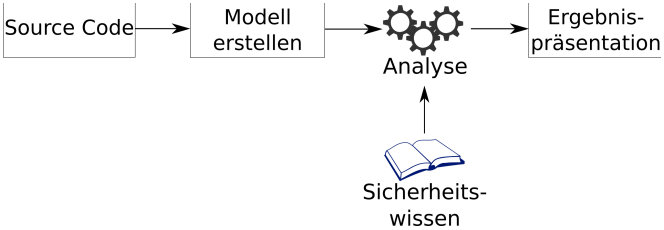
Secure Code Review – Apollo 11 Kommentare

```
CAF     CODE500         # ASTRONAUT: PLEASE CRANK THE
TC      BANKCALL        # SILLY THING AROUND
CADR    GOPERF1
TCF     GOTOP00H        # TERMINATE
TCF     P63SPOT3        # PROCEED SEE IF HE'S LYING

P63SPOT4    TC  BANKCALL    # ENTER     INITIALIZE LANDING RADAR
        CADR    SETPOS1

TC POSTJUMP    # OFF TO SEE THE WIZARD ...
        CADR    BURNBABY
```

(Vergleiche Apollo11 Github – `https://github.com/chrislgarry/Apollo-11/blob/`

Automatisierte statische Codeanalyse – Prozess

- Format der Parameter und angewendete Techniken variieren
- Durchführung anhand Source Code, Byte Code oder Binary möglich
- Sicherheitswissen beschreibt Regeln
  - Eingabequellen
  - Fehleranfällige Funktionen wie strcpy()



(Vergleiche Chess und West (2007))

Automatisierte statische Codeanalyse – Varianten

- Signaturbasiert
  - Suchen nach definierten Mustern wie sprintf, strcpy, ...
  - Tools: Flawfinder, RATS, ITS4
- Kontrollfluss-Analyse
  - Kontrollfluss: Ausführungsreihenfolge der einzelnen Instruktionen
  - Tools: Splint, MOPS (basierend auf Modell Checking)
- Datenfluss-Analyse
  - Zusammenhangs Erzeuger und Verbraucher von Daten
  - Vertrauenswürdigkeit der Quellen (Tainted)
  - Tools: LAPSE, Orizon

Fuzzing

Fuzzing – Input

(Vergleiche `http://www.google.com/googlebooks/chrome/small_10.html`)

Fuzzing

- Random Testing
- Automatisiertes Ausführen mit zufällig generierten oder vordefinierten Werten
- Testen der Datenfelder, Struktur und Ablauf
- Fuzzer für unterschiedliche Eingabeformate/Protokolle vorhanden
  - Dateien: Bilder, HTML, JSON, XML, DOC, SWF, ...
  - Protokolle: TCP/IP, SOAP, ...
  - ...
- Beispiel: „This problem occurs when the file size of the PNG file is 4,097 bytes or 4,098 bytes“ (`http://support.microsoft.com/kb/822071/en-us`)

(Vergleiche Sutton et al. (2007))

Fuzzing – Arten

- Unterscheidung nach **Input-Erstellung**
  - Generation-based Fuzzer → Generierung neuer Daten
  - Mutation-based Fuzzer → Abwandlung bestehender Daten
- Unterscheidung nach **Smartness**
  - Random Fuzzer → 'HdmxH&k ddadahtrr'
  - Template Fuzzer → 'GET /ex??ple.html'
  - Block Fuzzer → 'GET /example.hmtl'
  - Evolution-based Fuzzer → Lernen von Spezifika des Protokols durch Output wiederholter Abfragen
    - Umsetzung eines komplexeren Ablaufs, z.B. Handshake

Fuzzing – Ausführung

- Erzeugung und Ausführung automatisierbar
  - Große Anzahl von Datenvarianten möglich/erforderlich
  - Microsoft SDL verlangt 100.000 Test-Dateien bei Datei-Fuzzer
- Fehlererkennung
  - Veränderungen eines Systems: Prozessorleistung, Fehlereinträge in Log-Dateien, ...
  - Überwachen des Prozesses durch Debugger
  - Fehler der Applikation beobachten → segfault, Logikfehler, Abstürze, etc.
- Fehlerlokalisierung
  - Welcher Testfall hat Fehler verursacht?

Penetration Testing

If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat.

- Sun Tzu, The Art of War

(Vergleiche Tzu (2008))

Penetration Testing

- Spezielle Art von Sicherheitstests
  - Fehlersuche unter „realen“ Bedingungen
  - Ausnutzbarkeit der Schwachstellen zeigen → Beweis für Kunden/Management
  - Durchführung im/nahe am Betrieb
- Überprüfung unterschiedlicher Ebenen des Systems
  - z.B. Layer 1-7: technische Ebenen, Layer 8: Social Engineering
- Nahe an realen Bedrohungsszenarien
- Kostspielige Behebung gefundener Fehler (spät im Lifecycle)

Penetration Testing – Einführung

- Berücksichtigung des gesamten (Öko-)Systems (Betriebssystem, Datenbanken, Konfigurationen, Zusammenhänge, ...)
- Kreativität gefragt! (Thinking Out-Of-The-Box)
- Wichtig: Rechtliche Aspekte beachten!
  - Abstürze oder Schäden am System unter Test
  - Gesetzliche Bedingungen → Hacker Paragraph
  - Ist Auftraggeber berechtigt?
  - Dokumentation!
  - Handling von sensiblen Daten!
  - **„Get-Out-Of-Jail“**-Karte

Penetration Testing – Thinking Out-Of-The-Box

(Vergleiche `https://xkcd.com/538/`)

Penetration Testing – Rahmenbedingungen

- Was ist das Ziel des Tests?
- Welche Komponenten werden getestet?
- Welche Typen von Angreifern werden in Betracht gezogen?
- Abbruchbedingungen des Tests?
- Informationsbasis der Tester/der Getesteten?
- Verwendete Methodik und Techniken?

Penetration Testing – Phasen

- Versuch der Standardisierung von Prozessen zum Nachweis
- Grobe Phaseneinteilung:
  - Pre-Engagement
  - Intelligence Gathering
  - Threat Modelling
  - Vulnerability Analysis
  - Exploitation
  - Post-Exploitation
  - Reporting

(Vergleiche PTES Fork - `https://pentest-standard.readthedocs.io/en/latest/index.html`)

Ethical Hacking

- Suchen nach Schwachstellen ohne weitere Ausnutzung dieser
- Ethical Hacker je nach Definition auch als White- oder Gray-Hats bekannt
- Aufspüren von Schwachstellen OHNE Ausnutzen derselben für persönlichen Profit
- Beispielsweise durch Bug-Bounty-Programme
- Disclosure spielt große Rolle für Handling von Schwachstellen

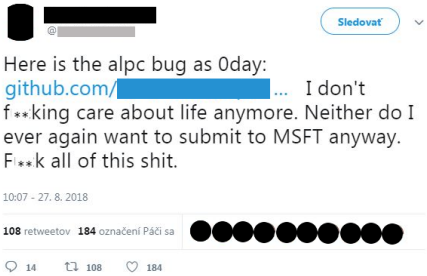
Vulnerability Disclosure

- **Responsible Disclosure**
  - Nicht sofort veröffentlichen! → Hersteller Zeit für Fix geben
  - Google Projekt Zero hat 90-Tage Deadline
  - Wenn Deadline vertreicht → Aufschub oder Full Disclosure
- **Full Disclosure**
  - Veröffentlichung so schnell als möglich!
  - Nutzer sollen selbes Wissen wie mögliche Angreifer besitzen!
  - Viele Experten unterstützen Full Disclosure, z.B.

Full disclosure - ... - is a damned good idea. Public scrutiny is the only reliable way to improve security, while secrecy only makes us less secure - **Bruce Schneier**

(Vergleiche `https://www.schneier.com/essays/archives/2007/01/schneier_full_disclo.html`)

Full Disclosure – Dropping a 0-Day



Responsible Disclosure – Example Timeline

Example: CVE-2018-9489 - Data Exposure via broadcasts in Android

- 2018-03-28: Initial report submitted to the vendor
- 2018-03-29: Initial response from the vendor received – issue being investigated
- 2018-05-02: Checking on status, response from vendor – issue still under investigation
- 2018-07-10: Response from vendor – issue still under investigation; pinged for a timeline
- 2018-07-12: Pinged the vendor regarding CVE assignment and disclosure plans
- 2018-07-13: Information about the fix provided by the vendor; follow-up communication
- 2018-07-19: Additional information provided to the vendor, response received
- 2018-08-09: Fix confirmed
- 2018-08-22: CVE assigned by the vendor, follow-up communication with the vendor
- 2018-08-23: Final version of the advisory provided to the vendor for review
- 2018-08-29: Public disclosure / advisory published; added information about Android forks
- 2018-09-05: Added Amazon's response

(Vergleiche `https://wwws.nightwatchcybersecurity.com/2018/08/29/sensitive-data-exposure-via-wifi-broadcasts-in-android-os-cve-2018-9489/`)

Schwachstellendatenbanken

- CERT (`http://www.cert.org/advisories/`)
- US-CERT Vulnerability Notes (`http://www.kb.cert.org/vuls/`)
- CVE (`http://cve.mitre.org/`)
- NIST – National Vulnerability Database (`http://nvd.nist.gov/`)
- Exploit DB (`https://www.exploit-db.com/`)
- CVE Details (`http://www.cvedetails.com/`)

Literatur 1/4

- Brian Chess und Gary McGraw. Static analysis for security.
  IEEE Security & Privacy, 2(6):76–79, November/Dezember 2004.
  ISSN 1540-7993.
  doi: 10.1109/MSP.2004.111
- Brian Chess und Jacob West. *Secure Programming with Static Analysis*.
  Addison-Wesley, 2007.
  ISBN 9780321424778
- Edsger W. Dijkstra. The humble programmer.
  Commun. ACM, 15(10):859–866, 1972.
  ISSN 0001-0782.
  doi: 10.1145/355604.361591

Literatur 2/4

- Elfriede Dustin, Jeff Rashka, und John Paul. *Software automatisch testen.*
  Springer, 2001.
  ISBN 3-540-67639-2
- Michael Howard und David LeBlanc. *Writing Secure Code, 2nd Edition.*
  Microsoft Press, 2002.
  ISBN 978-0-7356-9146-9
- Glenford J. Myers und Corey Sandler. *The Art of Software Testing.*
  John Wiley & Sons, 2004.
  ISBN 0471469122
- National Institute of Standards und Technology. Guideline on Network Security Testing. Recommendations of the National Institute of Standards and Technology, Oktober 2003.
  `http://csrc.nist.gov/publications/nistpubs/800-42/NIST-SP800-42.pdf`

Literatur 3/4

- Pete Herzog und Marta Barcelo. The Open Source Security Testing Methodology Manual, 2010
- Michael Sutton, Adam Greene, und Pedram Amini. *Fuzzing*.
  Addison-Wesley, 2007.
  ISBN 0-321-44611-9
- Herbert H. Thompson. Why security testing is hard.
  IEEE Security & Privacy Magazine, 1(4):83–86, 2003.
  ISSN 1540-7993.
  doi: 10.1109/MSECP.2003.1219078

Literatur 4/4

- Kenneth R. van Wyk. Adapting Penetration Testing for Software Development Purposes, Januar 2007.
  https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/penetration/655.html?branch=1&language=1
- D.W. Hoffmann. *Software-Qualität*.
  EXamen. press Series. Springer Berlin Heidelberg, 2008.
  ISBN 9783540763239.
  https://books.google.at/books?id=dUKsxRJGM9MC

Zusammenfassung

- Testen alleine reicht nicht aus → Sicherheit ist ein Prozess
- „Sicherheit kann nicht in ein System hineingetestet werden“ (Howard und LeBlanc (2002))
- Je später Tests stattfinden, desto kostenintensiver sind Fixes
- Fuzzing und Statische Codeanalyse als Test-Techniken
- Penetrationstests als spezielle Testart
- Full/Responsible Disclosure

Vielen Dank!

https://establishing-security.at/