DHBW Mosbach - IT Sicherheit – Sommersemester 2020

Sicherheitsanalyse, Fuzzing und Exploitation

Anweisungen & zu erbringende Leistungen

Da aufgrund der COVID-19 Pandemie keine schriftliche Klausur für die Vorlesung IT Sicherheit im Sommersemester 2020 stattfinden kann, wird stattdessen ein Projekt gestellt. Dieses besteht aus drei Teilen:
1. Es soll eine **Sicherheitsanalyse** des gegebenen Programms durchgeführt werden. Hierbei sollen Fähigkeiten demonstriert werden, welche durch die Vorlesungseinheiten von Herrn Schaller erlernt wurden.

2. Zu einer der gegebenen Software Pakete (siehe unten) soll ein **Fuzzer** geschrieben und betrieben werden. Hierzu wird AFL++ empfohlen.

3. Gefundene Schwachstellen sollen analysiert werden. Dabei soll beschrieben werden, ob und wie die Schwachstelle ausgenutzt werden kann. Hierbei soll Erlerntes aus dem Kapitel **Binary Exploitation** angewandt werden.

Als Abgabe wird ein **mindestens 5-seitiges PDF-Dokument** erwartet, das die Aufgaben 1 bis 3 beantwortet, sowie eine **Liste der Gruppenteilnehmenden mit individueller Bewertung der Leistungserbringung (siehe Benotung)**. Eventuell entstandener SourceCode kann als Zipdatei mitgeliefert werden.

Benotung

Es ist explizit **keine Punkteverteilung für die einzelnen Aufgaben** angegeben, da diese je nach zu betrachtender Software individuell sein kann. Es sollten jedoch zu jeder Aufgabe mindestens die Grundlagen beschrieben werden. Sollten bei Aufgabe 2 keine Schwachstellen gefunden werden, können diese manuell eingefügt werden um mit Aufgabe 3 fortzufahren.
Jede Gruppe erhält eine Gruppennote. Um die Leistungen der Studierenden besser **differenzieren** zu können wird mit Projektabgabe eine Liste der Gruppenteilnehmenden erwartet, die die Leistung der einzelnen mit den Indikatoren --, -, 0, + und ++ beschreibt. Idealerweise ist der Indikator bei allen 0, sodass alle gleich viel beigetragen haben.

Indikator	Notenabweichung von Gruppennote
--	+ 0,5
-	+ 0,25
0	0
+	- 0,25
++	- 0,5

Aufgabe 0: Vorbereitung

Das Projekt ist in **Gruppen von fünf bis sechs Studierenden** durchzuführen. Dabei soll eine Teamleiterin / ein Teamleiter gewählt werden, welcher die Kommunikation mit den Dozenten übernimmt. Die **TeamleiterInnen senden bis zum 02.06.2020 um 18.00 Uhr eine Liste der Teilnehmer sowie eine Liste mit drei Programmvorschlägen (siehe letzte Seite) nach Priorität sortiert an kschaller@ernw.de**. Nach Rückmeldung und Auswahl von einem Programm durch den Dozenten kann mit der Durchführung des Projektes gestartet werden. **Abgabe der Ausarbeitung ist der 30.06.2020 23:59 Uhr per E-Mail an kschaller@ernw.de.**

Aufgabe 1: Sicherheitsanalyse

(1) Beschreiben Sie, wie die ausgewählte Software in einer echten Umgebung betrieben wird. Zeigen Sie dabei mögliche Clients, Server und Kommunikationskanäle auf, die in einer solchen Umgebung verwendet werden (gerne in einem Schaubild). (2) Erörtern Sie im nächsten Schritt mögliche Bedrohungen und Schwachstellen, soweit Ihnen bekannt. (3) Wenden Sie dann die *Seven Sisters* Methodik auf die Umgebung an und beschreiben Sie jeweils, welche konkrete(n) Maßnahme(n) Sie in jeder Komponente implementieren würden und warum, und zeigen Sie auf wie das Risiko der vorher geschilderten Schwachstellen damit reduziert wird.

Aufgabe 2: Fuzzing

In diesem Teil des Projektes soll ein Fuzzer für das zugewiesene Programm geschrieben werden. Das Vorgehen unterscheidet sich natürlich und hängt von dem jeweiligen Programm und dessen Komponenten ab. Es kann zum Beispiel eine Bibliothek gefuzzt werden, welche von einem Programm verwendet wird oder das Programm selbst, wenn es in der Kommandozeile mit Parametern aufgerufen werden kann.

Es wird empfohlen, das Fuzzing mit AFL++ durchzuführen, so wie in der Vorlesung gelernt. Nach einer **Angriffsoberflächenanalyse** und Auswahl der zu untersuchenden Komponente soll ein **Korpus erstellt** werden. Hierbei können Inputs gesammelt, generiert sowie bereits mutiert werden. Dieser Korpus soll anschließend minimiert werden, um die Effizienz des Fuzzings zu steigern. Zusätzlich soll auf **mindestens 2 Kernen parallel** gefuzzt werden. Optimierungen des Fuzzers wie das Verzögern des **Fork Servers** oder Verwenden des **Persistent Modes** und **Shared Memory** sind gerne gesehen, falls anwendbar. Ebenfalls können **Sanitizers** verwenden werden wie der Address Sanitizer oder der Undefinded Behaviour Sanitizer.

Die Crashes können mit **crashwalk** zusammengefasst werden. Sollte der Fuzzer nach mehreren Tagen und Iterationen keine Crashes finden, können nach Rücksprache selbst Schwachstellen eingefügt werden um mit Aufgabe 3 fortzufahren.

Wenn zusätzliche Ressourcen wie ein Server zum Fuzzen benötigt werden sollten, kann dieser auf Anfrage bereitgestellt werden.

Aufgabe 3: Binary Exploitation

In diesem Schritt geht es darum, die gefundenen Crashes zu verstehen und im besten Fall zu exploiten. Da die meisten Crashes wahrscheinlich keine simplen Stack Buffer Overflows sein werden, wird ein vollständiger Exploit nicht erwartet. Gegebenenfalls können, wie in Teil 2 beschrieben, auch eigene Schwachstellen in den Source Code der Software eingebaut werden. Vorhandene Crashes sollten dennoch so gut es geht erforscht und dokumentiert werden. Welche Schutzmaßnahmen (z.b. DEP) sind für die Software aktiv? Es soll gezeigt werden, was den Crash auslöst und welche Speicherbereiche oder Register man ggf. kontrolliert.

Vorschläge für Software

- GoldenCheetah
  - Performance Software for Cyclists, Runners and Triathletes
  - Parses many File Formats (GPX, SRM, TCX...)
  - https://github.com/GoldenCheetah/GoldenCheetah
- FIT SDK
  - Flexible and Interoperable Data Transfer
  - Used in lots of Garmin Devices
  - https://www.thisisant.com/resources/fit-sdk-beta
- Weather Software
  - General Regularly-distributed Information in Binary form
  - Used in meteorology to store weather forecasts
  - https://www.zygrib.org/
- Digital Audio Broadcasting (DAB/DAB+)
  - Ensemble Transport Interface
  - ODR-DabMux: https://github.com/Opendigitalradio/ODR-DabMux
  - ODR-DabMod: https://github.com/Opendigitalradio/ODR-DabMod
  - dablin: https://github.com/Opendigitalradio/dablin
  - etisnoop: https://github.com/Opendigitalradio/etisnoop/
  - odr-edilib: http://git.mpb.li/git/odr-edilib/
- Bioinformatics Software
  - E.g.: Ugene
  - Gene Classification, Creating Protein Sequences, ...
  - https://ugene.net

- GRASS GIS
  - Geospatial data management, vector and raster manipulation
  - https://grass.osgeo.org/
- BSP File Format Fuzzing
  - Map file used in games like Quake and Counter-Strike
  - Open Source Implementations and Releases from Vendors
  - https://github.com/id-Software/Quake-2-Tools
  - https://github.com/id-Software/Quake-III-Arena
  - https://github.com/ioquake/ioq3
- eigener Vorschlag