Entwicklungsprojekt WS 2019/20

Nebenperspektive Weaving the Web

In der folgenden Ausarbeitung wird die Nebenperspektive Weaving the Web, wie sie im Entwicklungsprojekt beahandelt wurde, erläutert.

Das System

Das Projekt handelt von der Entwicklung eines Phishing Trainingscenters. Hierbei sollen angemeldete Nutzer immer wieder simulierten Angriffen aus dem Bereich EMail und Werbe Phishings ausgesetzt werden, welche sich mit der Zeit an die Fähigkeiten des Nutzers anpassen. Dieser kann zudem seine Erfolge und Misserfolge überwachen und seine eigene Entwicklung im Auge behalten.

Die Komponenten

Nutzerverwaltung

Die Nutzerverwaltung (welche in unserem Prototyp recht klein ausgefallen ist und nur aus dem entsprechenden DB Modell besteht und in einer realen Entwicklung ein gesonderter Service wäre), identifiziert Nutzer, verbindet diese mit Angriffen und verwaltet auch Rollen und Gruppen eines Nutzers.

Kommuniziert mit:

- Datenbank
- Angriffsgenerator
- Angreifern
- Dashboard

Datenbank

Wegen der einfachen Umsetzung haben wir in unserem Beispiel auf eine SQLite DB gesetzt, da diese sich leicht nutzen lässt. In einer realen Umgebung würde dies entweder gegen eine zentrale SQL Datenbank getauscht werden, oder noch eher gegen eine NoSQL DB (z.B. MongoDB), da die unterschiedlichen Angreifer verschiedene Daten speichern, welche nicht zwingend relational darstellbar sind.

Kommuniziert mit:

- Angriffsgenerator
- Nutzerverwaltung
- Angreifern
- Dashboard

Angriffsgenerator

Der Angriffsgenerator ist dafür zuständig sämtliche möglichen Angreifer zu aktivieren und eventuell in (nicht)zyklischen Abständen diese aufzufordern einen Angriff gegen einen Nutzer zu starten. Aktuell werden

dafür ledeglich direkte Funktionsaufrüfe genutzt, doch im realen Umfeld würden diese gegen z.B. Message Queueing getauscht werden.

Kommuniziert mit:

- Angreifern
- Nutzerverwaltung
- Datenbank

Angreifer

Ein Angreifer ist dafür zuständig eine gewisse Klasse an Angriffen (Mail/Web/...) durchzuführen. In einer realen Umsetzung ließen sich diese in eigene Microservices auslagern.

Kommuniziert mit:

- Nutzerverwaltung
- Angriffsgenerator
- Datenbank

Mail Angreifer

Dieser Angreifer erledigt Angriffe via Mail. Diese umfassen zum Beispiel Spam Mails oder Phishing Angriffe (gezielt/ungezielt). Dieser ist eine Spezifizierung des Angreifers.

Kommuniziert zusätzlich mit:

- Nutzer Mail Dienst
- Datenbank

Web Angreifer

Dieser Angreifer erledigt Angriffe via Web Anfragen. Diese umfassen zum Beispiel ersetzte Werbeanzeigen oder Phishing Seiten (gezielt/ungezielt). Dieser ist eine Spezifizierung des Angreifers.

```
Anmerkung:
Dank der neueren Chrome Updates Ende 2019, welche den Umgang von http Inhalten im
https Kontext und der Möglichkeiten von PlugIns verändert haben, ist der Einsatz
dieses Angreifers erheblich erschwert worden. Dies benötigt nun ein extra
angelegtes root Zertifikat auf der Nutzermaschine.
```

Kommuniziert zusätzlich mit:

- Nutzer Browser
- Datenbank

DNS Dienst

Für die Nutzung des Web Angreifers ist ein modifizierbarer DNS Dienst nötig, um Umleitungen bestimmter Domains vornehmen zu können (z.B. Google Ads). Auf diese Weise kann der Web Angreifer reale Werbeanzeigen in echten Websites ersetzen und somit realistischere Angriffe erzielen. Der Dienst kann unabhängig von allen anderen Komponenten agieren.

Kommuniziert mit:

- Nutzer Browser

Dashboard

Das Dashboard gibt die Möglichkeit für Nutzer Gruppen und Rollen zu verwalten, die eigene Leistung zu beobachten und vergangene Angriffe einzusehen. Zudem ist es möglich bewusst Angriffe zu generieren. Hierfür wird einseitig in Richtung Angriffsgenerator kommuniziert.

Kommuniziert mit:

- Angriffsgenerator
- Nutzer Browser
- Nutzerverwaltung
- Datenbank

Interaktionen Synchron / Asynchron

Aus unterschiedlichsten Gründen (Entwicklungsaufwand, Wissensstand / fehlender Erfahrung eines Teammitglieds / ...) wurde im Prototypen die Kommunikation rein synchron aufgesetzt.

Die Komponentenstruktur wurde jedoch so erdacht, dass sich in einer realen Umsetzung sämtliche interne Kommunikation asynchron gestalten lässt. Auf diese weise ließe sich z.B. in einem Firmenumwelt ein existierendes LDAP einbinden, die Nutzerverwaltung als Dienst von den Angreifern trennen und auch die Datenbank auslagern. Hierbei würde als weitere Komponente ein Message Broker hinzukommen, welcher mit allen internen Komponenten interagieren würde.

Protokolle

Hierbei würden wir zur internen Kommunikation aktuell RabbitMQ oder MQTT enpfehlen, da es passieren kann, dass Binärdaten im System verschickt werden müssen (z.B. Bilder aus Webangriffen). Zudem gibt es Events (z.B. das Starten eines Angriffs auf einen Nutzer), welche auch indirekte Auswirkungen auf andere Komponenten haben können. So sorgt zum Beispiel ein manuell erzeugter Angriff dafür, dass der Angriffsgenerator nicht direkt danach noch einen Angriff generiert.

Auch die Kommunikation nach außen wurde so gut es geht asynchron gestaltet. So kann der Mail Angreifer dank IMAP Protokoll über eingehende Mails informiert werden und somit asynchron agieren. Die Kommunikation mit dem Browser passiert naturgemäß über HTTP.

Räumliche Teilung

Im Prototyp wurde keine räumliche Teilung vorgenommen. In der beschriebenen Realumsetzung wäre allerdings eine Teilung auf beliebig viele Rechner möglich und auch einzelne Dienste lassen sich verteilt

betreiben.

Daten

Grundsätzlich werden zwischen Systemkomponenten nur Daten der beteiligten Systeme ausgetauscht (z.B. die Nutzerverwaltung kann genaue Daten über einen Nutzer an einen Angreifer senden). Sollten Daten aus anderen Systemen benötigt werden (z.B. Dashboard -> Angreifer "greife Nutzer X an"), so werden nur Refferenzen auf diese Daten ausgetauscht (im Beispiel die Nutzer ID). Auf diese Weise kann jede Komponente stets kontrollieren, wer welche Daten abrufen möchte und zudem kann die Kommunikation weiterführend optimiert werden (z.B. ein Angreifer braucht im Zweifel alle Daten eines Nutzers, während Angriffsgenerator nicht die Mail Adresse benötigt).

Sprachen

Aufgrund der Bekanntheit im Team, des framework/library supports, der systemunabhängigen Einsatzmöglichkeit und der intern asynchron aufgebauten Struktur wurde sich bei der Umsetzung des Prototypens komplett für JavaScript mit nodeJS entschieden (Frontend mit HTML/CSS). Aufgrund der bereits angesprohenen Möglichkeit der Aufteilung der Systeme wäre es nicht schwer einzelne Komponenten neu in anderen Sprachen zu entwickeln, doch wir sehen hierfür keine Nötigkeit. Ledeglich auf ein Superset wie Typescript könnte man für eine reale Umsetzung umsteigen, um den Entwicklungsfluss weiterführed zu vereinfachen. Allerdings war TS nie Teil des Studiums und hätte wegen mangelnder Erfahtung die Entwicklung deutlich verlangsamt.

Zudem war es auf diese Weise nicht nötig Build Tools in die Entwicklung des Prototypens einzubringen.

Es wurde allerdings ein sehr moderner ES6 Stil in der Entwicklung verwendet, welcher bereits ES6 Module statt CommonJS Module verwendet, welches von den neuen nodeJS Versionen unterstützt wird.

Middlefwaresysteme / Frameworks + Typsystem

Abseits der gängigen Bibliotheken für einen möglichen Message Broker würden wir in einer realen Umsetzung für den Datenbankzugriff entweder das aktuell benutzte Sequelize empfehlen (unterstützt auch MongoDB), oder einen wechsel auf Mongoose oder ein vergleichbares ORM für NoSQL DBs vorschlagen. Die wahl dieses ORMs war vor allem durch gute Erfahrungen in der Vergangenheit und die gute Integration von Seeds, Migrationen und Modellen begründet.

Hieran sieht man auch schon unser Typsystem, welches stringent auf ES6 Klassen aufbaut und dem Sequelize Modelle zugrunde liegen.

Neben dem bereits angesprochenen Sequelize ORM nutzen wir im Prototypen zudem Express mit seinen Middlewares im Verbund mit der Templating Engine Nunjucks. Zudem kommen einige Librarys für z.B. imap und smtp Mailverkehr.

Insgesamt kann man hier also von einem sehr "einfachen" Standardsetup sprechen, welches keine besonderen Probleme bereiten sollte. In einer realen Umsetzung wäre dennoch ein Wechsel auf z.B. NestJS als Framework für Webanwendungen für einzelne Komponenten wie das Dashboard zu empfehlen, um die Komplexität mehr einzugrenzen und klare Strukturen bereits durch ein Framework zu erhalten.

Da das Frontent im Prototypen so gut wie nicht angefasst wurde, wäre hier gerade für das Dashboard eine Umsetzung in z.B. WebComponents, Preact oder Vue denkbar.

Persistente Speicherug

Daten werden im Prototypen in SQLite Datenbanken persistent abgelegt. Hierfür wird wie erwähnt das ORM Sequelize genutzt, welches durch eine einzige Einstellung auf z.B. MySQL umgestellt werden kann, Somit ist die gesamtgestaltung problemlos auch in einem Produktionsumfeld nutzbar. Sollte man allerdings wie empfohlen auf Mongoose umsteigen, so käme dort nochmals Arbeit auf. Dies wurde noch nicht getan, weil dieser "Fehler" erst spät in der Entwicklung sich als tatsächliches Problem herausgestellt hat und auch anders (nicht ganz so schön) gelöst werden konnte.

Anforderungen an den Nutzer

Für eine minimale Nutzungsmöglichkeit benötigt der Nutzer ledeglich einen WebBrowser auf einem beliebigen Gerät. Damit kann er sich generierte Werbeattacken anzeigen lassen. Sollte er zudem ein Mail Konto besitzen, so schaltet sich (nach entsprechnder Eingabe) der MailAttacker für diesen Nutzer aktiv und sollte ein Root Zertifikat eingespeist worden sein (z.B. für einen Firmenproxy), so kann er den vollen Produktumfang genießen.

Dienste im Web

Unser Projekt kann mehr oder weniger autag agieren, profitiert allerdings von der Integration in externe Dienste. Hierbei handelt sich allerdings meist nicht um öffentliche Webdienste, sondern z.B. Firmeninterne Dienste, wie ein Proxy mit DNS, ein Mailserver. Zudem wäre es langfristig denkbar z.B. existierende Datenbanken über Angriffe einzubinden, um realistischere Angriffe generieren zu können. Dabei sollten allerdings niemals Daten aus dem System heraus nach ausßen gelangen.

Da alle Angriffe basierend auf Tempates erstellt werden, lassen sich allerdings problemlos anonyme Analysen über Angriffe und deren Erfolg erstellen. Hierbei muss nur noch die Nutzer ID anonymisiert werden und potenziell Metadaten entfernt oder unkenntlich gemacht werden.

Wesentliche Funktionalitäten

Um den Prototypen demonstrieren zu können, sollten mindestens folgende Komponenten Aktiv sein:

- Die Nutzerverwaltung
- Die Datenbank
- Der Angriffsgenerator
- Mindestens ein Angreifer
- Zwei aktive Testnutzer

Mit dieser Konstellation lässt sich demonstrieren, wie ein Nutzer zu beginn eine Bewertung im Dashboard hat, ein Angriff generiert wird, der Nutzer auf diesen Positiv reagiert, daraufhin eine Verbesserung im Dashboard zu erkennen ist, er auf einen weiteren generierten Angriff, der nun schwieriger ist, negativ reagiert und somit eine Verschlechterung im Dashboard sieht. Zwischendrin lässt sich das System auch neustarten. Danach wird sich mit einem zweiten Nutzer angemeldet, der dann das Dashboard aufruft und nicht die gerade getätigten Angriffe sieht.

Auf diese weise werden folgende Eigenschaften demonstiert:

- Das System kann Angriffe generieren
- Das System geht auf die Reaktionen des Nutzers ein

- Die Bewertung funktioniert
- Die eigene Überwachung lässt sich nachvollziehen
- Die Kommunikation im System funktioniert
- Die persistente Speicherung ist vorhanden
- Die Trennung der Nutzer funktioniert.

Somit ist die Grundfunktionalität im Prototypen enthalten und ein Einsatz mit einer Testgruppe möglich.

Anmerkungen

Während der Entwicklung des Projekts wurde der Schwerpunkt auf die Hauptperspektive gelegt und somit die Nebenperspektive etwas nach hinten gestellt. Bei der Konzeption wurde auf saubere Arbeit geachtet (Modularisierung der Komponenten, Trennbarkeit, Bedacht von realen Umsetzungsmöglichkeiten), auch wenn diese in der Implementierung des Prototypen potentiell nicht umgesetzt wurden, um Zeit und Entwicklungsaufwand zu sparen.

Auf diese Weise konnte Zeit für eine kleine Nutzerstudie im realen Einsatz gewonnen werden, welches deutlich darstellte, dass die aktuelle Form des Prototypens zeigt, dass ein solches Produkt definitiv sinnvollen Einsatz finden würde.