

ПОРІВНЯЛЬНИЙ АНАЛІЗ ФАЗЗІНГОВИХ ФРЕЙМВОРКІВ

Пропонується огляд та порівняльний аналіз інтегрованих середовищ фаззінгу, відомих як фаззінгові фреймворки. Встановлені основні характеристики таких фреймворків. Результати даної роботи можуть бути використані для здійснення обґрунтованого вибору фаззінгового фреймворку для розв'язання задач підвищення захищеності програмного забезпечення.

Ключові слова: фаззінг, фаззінговий фреймворк, інтегроване середовище фаззінгу, виявлення уразливостей, захищеність програмного забезпечення.

Вступ. Існують різні підходи до підвищення захищеності програмного забезпечення. Серед них особливу групу становлять методи виявлення уразливостей програм, які на даний момент поділяються на три основні напрямки: виявлення уразливостей за допомогою аналізу початкових текстів програм; виявлення уразливостей за допомогою аналізу двійкових кодів програм; виявлення уразливостей за допомогою фаззінгу [1]. Останнім часом найбільш перспективним вважається виявлення уразливостей за допомогою фаззінгу [2].

Програмні засоби, які реалізують фаззінг, називають фаззерами.

Аналіз досліджень і публікацій, присвячених виявленню уразливостей за допомогою фаззінгу, в т.ч. [1–5], показав, що існує ціла низка спеціалізованих фаззерів, які працюють з багатьма відомими та задокументованими мережевими протоколами і файловими форматами. Незважаючи на те, що застосування цих фаззерів досить ефективно при роботі з цілою низкою популярних застосунків, часто буває необхідним більш індивідуальний підхід — для ретельного фаззінгу користувацьких протоколів та протоколів, які раніше не досліджувалися. Саме в цих випадках надзвичайно корисними є так звані інтегровані середовища фаззінгу [2], або фаззінгові фреймворки (англ. *fuzzing frameworks*).

Фаззінгові фреймворки (ФФ), як правило, є набором підпрограм, які можуть використовуватися для швидкого написання фаззера [3]. Крім того, фреймворк може стати гарною основою для розробки нової фаззінгової системи виявлення уразливостей.

Постановка завдання. Проте, засобів, які позиціонують себе як ФФ чимало, і усі вони мають різноманітні характеристики, свої переваги і недоліки. Отже, виникає актуальна потреба провести порівняльний аналіз ФФ, що дозволить здійснювати обґрунтований вибір фреймворку для розв'язання задач підвищення захищеності певного програмного забезпечення. Зважаючи на це, завданням даної роботи є порівняльний аналіз низки ФФ та виявлення їх основних характеристик. При цьому не ставиться за завдання охопити усі системи, які намагаються позиціонувати себе як фреймворки. Принципами відбору розглянутих далі інтегрованих середовищ є їх досконалість та функціональна оснащеність.

Виклад основного матеріалу досліджень. Фаззінгові фреймворки часто використовуються для сприяння легкому і швидкому створенню «одноразових» фаззерів. Основна ідея цих видів засобів — повторне використання коду [3]. Фреймворк повинен надавати засоби для максимізації можливості повторного використання коду. Розроблені компоненти повинні бути легкодоступними для творців майбутніх проектів. При правильній реалізації ця концепція забезпечить еволюцію фаззера — чим більше він буде використовуватися, тим «розумнішим» стане [2].

Фаззери можуть мати різні рівні знання протоколу. Деякі фаззери реалізовані як інтегровані середовища, які забезпечують кінцевого користувача платформою для створення фаззінгових випробувань для довільних протоколів. Фаззінгові фреймворки, як правило, вимагають значних часових та ресурсних витрат на моделювання випробувань для нового інтерфейсу, і якщо інтегроване середовище не пропонує готових вхідних даних для спільних структур та елементів, ефективного випробування також вимагає значного досвіду у

проектуванні вхідних даних, які здатні викликати помилки в інтерфейсі, що випробується. Деякі ФФ інтегрують створені користувачами тестові модулі назад до платформи, в результаті чого нові випробування в межах досяжності інших користувачів, але у цілому ФФ вимагають нових тестів, щоб завжди бути реалізованими «з нуля». Ці чинники можуть обмежувати доступність, зручність використання та застосовність фреймворків [3].

Гарний ФФ повинен запобігати виконанню рутинних завдань та мінімізувати їх кількість. Для того щоб полегшити проходження перших етапів моделювання протоколів, в деякі фреймворки вбудовані програми, що конвертують перехоплений мережевий трафік у формат, який використовує дане інтегроване середовище [2].

Більшість ФФ реалізує методи генерації псевдовипадкових даних. Розвинуті фреймворки пішли далі і напрацьовують цілий набір евристик атаки. Евристика атаки — це ніщо інше як збережена послідовність даних, яка призвела до програмної помилки. Реалізація циклу через кінцевий перелік подібних тестових випадків перед тим, як приступити до генерації випадкових даних, може зекономити час [2].

За результатами досліджень пропонується для порівняльного аналізу використати наступні основні характеристики ФФ: мова реалізації; платформа (операційна система), на якій працює фреймворк; мова розробки фаззерів; автоматизація формування формату протоколу; наявність наборів евристик атак; можливість повторного використання коду; доступність використання. Результати порівняльного аналізу ФФ приведені у табл. 1.

Antiparser — програмний інтерфейс, написаний мовою Python і призначений для використання при створенні випадкових даних, особливо при розробці фаззерів. Це інтегроване середовище може бути використане для розробки фаззерів, які можна запускати на різних платформах, оскільки фреймворк залежить виключно від доступності інтерпретатора мови Python. Незважаючи на те, що це інтегроване середовище просте та має певні переваги при генерації простих фаззерів, воно буде недоречне при розв'язанні більш складних задач [2].

Dfuz — ФФ, написаний мовою C, з відкритим початковим кодом. Цей фреймворк був використаний при розкритті різноманітних уразливостей у програмному забезпеченні від таких компаній як Microsoft, Ipswitch та RealNetworks [2]. Dfuz був призначений для використання в операційних системах Unix/Linux та використовує власну мову для розробки нових фаззерів. Даний ФФ має простий та інтуїтивно зрозумілий дизайн, що робить його гарним учбовим прикладом улаштування фреймворку. Dfuz дозволяє в певному ступені повторне використання коду, але навіть і близько не в тому ступені, в якому це пропонують інші фреймворки, наприклад Reach. Невистачає також інтелектуального набору евристик атаки [2].

SPIKE написаний на мові C і є, мабуть, найбільш популярним та визнаним ФФ [2]. Його початкові коди відкриті. Це інтегроване середовище надає користувачу програмний інтерфейс, що дозволяє швидко та ефективно розробляти фаззери мережевих протоколів [1; 2]. Остання на даний час версія 2.9 цього фреймворку містить набір, що нараховує майже 700 евристик атаки [2]. SPIKE застосував новаторську у свій час технологію представлення мережевих протоколів, а відтак і їх фаззінгу. Структури даних протоколу розбиваються та представляються у вигляді блоків (що також називаються «spike»), у яких містяться і двійкові дані, і дані про розмір блоку. Подання протоколу у вигляді блоків дозволяє створювати в абстрактному вигляді різноманітні шари протоколу, і автоматично визначати його розміри [2]. Згодом ціла низка ФФ перейняли у SPIKE цю технологію блокового методу фаззінгу [2]. Безсумнівними недоліками SPIKE є такі: 1) навіть для здійснення незначних змін фреймворку (наприклад, додавання нових рядків фаззінгу) потрібна повторна компіляція; 2) повторне використання коду вже розроблених фаззерів полягає у його копіюванні та вставленні вручну [2].

Reach є міжплатформним ФФ з відкритим початковим кодом, написаним на мові Python. У порівнянні з іншими доступними фреймворками Reach, мабуть, володіє найгнучкішою архітектурою, яка найбільш сприятлива для повторного використання коду [2; 3]. Дане інтегроване середовище має цілу низку базових компонентів, необхідних для створення нових фаззерів, включаючи генератори, перетворювачі, протоколи, видавці та групи [2]. Генератори відповідають за генерацію даних — від простих рядків до багатопланових двійкових повідомлень. Для спрощення процесу генерації складних типів даних генератори можуть об'єднуватися у ланцюжки. Перетворювачі змінюють дані та можуть об'єднуватися у ланцюжок і приєднуватися до генератору. Так звані видавці забезпечують транспортування згенерованих даних через протокол. Абстрагування процесів генерації та перетворення даних, видавців в окремі об'єкти забезпечує простоту повторного використання коду вже розроблених фаззерів. Групи містять один або більше генераторів і є апаратом, що забезпечує проходження через усі значення, які може породити генератор [2]. Виходячи з теоретичних засад, Reach є досить ефективним фреймворком.

Таблиця 1

Порівняльний аналіз фаззінгових фреймворків

ФФ	Мова реалізації	Платформи	Мова розробки фаззерів	Автоматизація формування формату протоколу	Наявність евристик атак	Повторне використання коду	Доступність використання
Antiparser	Python	Будь-яка, що підтримує мову Python	Python	—	—	ручне	просте використання; дуже бідна документація
Dfuz	C	Unix/Linux	власна скриптова мова	емуляція FTP, POP3, Telnet, SMB	—	обмежене	просте використання
SPIKE	C	Unix/Linux	C	блокове подання	майже 700	тільки шляхом копіювання вручну	несистематизована та заплутана документація
Reach	Python	Будь-яка, що підтримує мову Python	XML	моделювання субкомпонентів протоколу	—	гнучке	доступна документація
GPF	C	Unix/Linux	власна скриптова мова	мутація перехопленого трафіку та двійкових протоколів	—	—	складне використання
Autodafé	C	Unix/Linux	C	блокове подання	+	ускладнене	докладна документація [4]
Fuzzled	Perl	Будь-яка, що підтримує мову Perl	скриптова мова	моделювання певних протоколів	+	можливе	дуже бідна документація
Sulley	Python	Будь-яка, що підтримує мову Python	Python	блокове подання	+	гнучке	докладна документація [2]

GPF був розроблений для використання на платформі Unix і має відкритий початковий код. Основна перевага цього ФФ полягає у мінімальних витратах, необхідних для початку розробки та запуску фаззера. У цілому, GPF є цінним інструментом завдяки його гнучкості та здатності до розширення. Здатність до автоматичної обробки та фаззінгу протоколів ASCII є дуже потужною і виділяє його серед інших інтегрованих середовищ [2; 3].

Autodafé називають фреймворком SPIKE наступного покоління, який може бути використаний для фаззінгу мережевих протоколів та форматів файлу [2]. Ця система призначена для використання на платформі Unix, написана на мові C і поширюється з відкритим початковим кодом. Найбільш вражаюча характеристика цього середовища — налагоджувач. Для внесення змін у цей фреймворк необхідно повторна компіляція [2; 4].

Одним з наймолодших ФФ є Fuzzled, розроблений на мові Perl. Fuzzled є подібним до Peach у тому, що допоміжні функції дозволяють розробляти широкий спектр засобів фаззінгу. Fuzzled містить код, який допомагає з різними евристичними, побудовою певних протоколів та реалізує інші функції. Зокрема, він підтримує NNTP, SMTP, IMAP та інші протоколи [3; 5].

Sulley — це інтегроване середовище розробки фаззерів та здійснення фаззінгової перевірки, що складається з великої кількості розширюваних компонентів. На думку авторів праці [2] фреймворк Sulley перевершує за своїми можливостями усі раніше випущені технології фаззінгу. Мета даного ФФ спростити не тільки подання даних, але також передачу даних та моніторинг об'єкту. Також Sulley виявляє, відслідковує та класифікує виявленні помилки. Цей засіб здатний виявляти те, яка унікальна послідовність тестових випадків призвела до появи помилок.

Висновки. Таким чином, було запропоновано для порівняльного аналізу ФФ використати наступні їх основні характеристики: мова реалізації; платформа; мова розробки фаззерів; автоматизація формування формату протоколу; наявність наборів евристик атак; можливість повторного використання коду; доступність використання. Був проведений аналіз наступних фреймворків: Antiparser, Dfuz, SPIKE, Peach, GPF, Autodafé, Fuzzled, Sulley. Одним з найрозвинутіших інтегрованих середовищ виявився Sulley.

Результати цієї роботи можуть бути використані для здійснення обґрунтованого вибору фреймворку для розв'язання задач підвищення захищеності програмного забезпечення, зокрема для швидкої розробки фаззерів або побудови нової фаззінгової системи виявлення уразливостей.

Література

1. *Козиол Дж.* Искусство взлома и защиты систем = The Shellcoder's Handbook / Джек Козиол, Дэвид Личфилд, Дэйв Эйтэл, Крис Энли, Синан Эрен, Нил Мехта, Рили Хассель. — Пер. с англ. — СПб. : Питер, 2006. — 416 с. — ISBN 5-469-01233-6.
2. *Саттон М.* Fuzzing: Исследование уязвимостей методом грубой силы / Майкл Саттон, Адам Грин, Педрам Амини. — Пер. с англ. — СПб. : Символ-Плюс, 2009. — 560 с. — ISBN 978-5-93286-147-9.
3. *Takanen A.* Fuzzing for Software Security Testing and Quality Assurance / Ari Takanen, Jared DeMott, Charlie Miller. — Norwood, MA, USA: Artech House, 2008. — 287 p. — ISBN 978-1-59693-214-2.
4. *Vuagnoux M.* Autodafé: an Act of Software Torture [Electronic resource] / Martin Vuagnoux // 22nd Chaos Communication Congress. — Electronic data. — Hamburg, Germany : Chaos Computer Club e. V., 2005. — Mode of access: World Wide Web. — URL: <http://events.ccc.de/congress/2005/fahrplan/events/606.en.html>. — Title from screen. — Description based on version dated 2007 January 15.
5. *Brown T.* Writing a fuzzer using the Fuzzled framework [Electronic resource] / Tim Brown // Nth Dimension. — Electronic data. — London : Portcullis Computer Security Ltd., 2008. — Mode of access: World Wide Web. — URL: <http://www.nth-dimension.org.uk/pub/WAFUTFF.pdf>. — Title from screen. — Description based on version dated 2008 February 24.

Надійшла: 20.03.11

Рецензент: д.т.н., проф. Юдін О.К.