УДК 004.056:004.415.532(045)

К.П. Ануфрієнко, кандидат технічних наук НАУ, асистент
(Національний авіаційний університет, Україна)

ОГЛЯД ІНТЕГРОВАНИХ СЕРЕДОВИЩ ФАЗЗІНГУ

Пропонується огляд та порівняльний аналіз інтегрованих середовищ фаззінгу. Встановлені основні характеристики таких середовищ. Результати даної роботи можуть бути використані для здійснення обґрунтованого вибору інтегрованого середовища фаззінгу для розв'язання задач підвищення захищеності 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].

Фаззінг (fuzzing) — це метод виявлення уразливостей у програмному забезпеченні за допомогою передачі йому різноманітних несподіваних вхідних даних та моніторингу винятків, тобто проблемних реакції під час виконання. Звичайно це автоматизований або напівавтоматизований процес. Це досить загальне визначення, але воно висвітлює основну концепцію 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 та файловими форматами. Незважаючи на те, що застосування цих фаззерів досить ефективне 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 (ІСФ) [2], або фреймворки (англ. fuzzing frameworks).

Інтегровані середовища фаззінгу, як правило, є набором підпрограм, які можуть бути використані для швидкого написання фаззеру [3, с. 146]. Крім того, фреймворк фаззінгу може стати гарною основою для розробки нової фаззінгової системи виявлення уразливостей.

Постановка завдання. Проте засобів, які позиціонують себе як ІСФ чимало, і усі вони мають різноманітні характеристики, свої переваги і недоліки. Отже, виникає актуальна потреба провести порівняльний аналіз ІСФ, що дозволить здійснювати обґрунтований вибір фреймворку для розв'язання задач підвищення захищеності певного програмного забезпечення. Тому завданням даної роботи є порівняльний аналіз низки ІСФ 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. Інтегровані середовища фаззінгу часто використовуються для сприяння легкому і швидкому створенню «одноразових» фаззерів. Основна ідея цих видів засобів — повторне використання коду [3, с. 146]. Фреймворк фаззінгу 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 фаззеру — чим більше він буде використовуватися, тим «розумнішим» стане [2].

Фаззери можуть мати різні рівні знання протоколу. Деякі фаззери реалізовані як ІСФ, які забезпечують кінцевого користувача платформою для створення фаззінгових випробувань для довільних протоколів. Фреймворки фаззінгу, як правило, вимагають значних часо-

вих та ресурсних витрат для моделювання випробувань для нового інтерфейсу, і якщо інтегроване середовище не пропонує готових вхідних даних для спільних структур та елементів, ефективне випробування також вимагає значного досвіду у проектуванні вхідних даних, які здатні викликати помилки в інтерфейсі, який випробується. Деякі ІСФ інтегрують створені користувачами тестові модулі назад до платформи, в результаті чого нові випробування в межах досяжності інших користувачів, але у цілому інтегровані середовища фаззінгу вимагають нових тестів, щоб завжди бути реалізованими з нуля. Ці чинники можуть обмежувати доступність, зручність використання та застосовність ІСФ [3, с. 31].

Гарне ІСФ повинне 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 ІСФ 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 контрольних прикладів перед тим, як приступити до генерації випадкових даних, може зекономити час [2].

За результатами досліджень пропонується для порівняльного аналізу використати наступні основні характеристики ІСФ: мова реалізаці; 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 фреймворків фаззінгу 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, с. 371].

Dfuz — ІСФ, написане 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, с. 373]. Dfuz був призначений для використання в операційних системах Unix/Linux та використовує власну мову для розробки нових фаззерів. Дане ІСФ 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 Peach. Невистачає також інтелектуального набору евристик атаки [2, с. 377].

SPIKE написаний на мові C і є, мабуть, найбільш популярним та визнаним фреймворком фаззінгу [2, с. 378]. Його початкові коди відкриті. Це ІСФ 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, с. 379].

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, с. 378]. Згодом ціла низка ІСФ перейняли у SPIKE цю технологію блокового методу фаззінгу [2, с. 381].

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, с. 381].

Таблиця 1. Порівняльний аналіз інтегрованих середовищ фаззінгу

ІСФ	Мова реалізації	Платформи	Мова розробки фаззерів	Автоматизація формування формату протоколу	Наявність евристик атак	Повторне використання коду	Доступність використання
Antiparser	Python	Будь-яка, яка 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
Peach	Python	Будь-яка, яка 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, яка підтримує мову Perl	скриптова мова	моделювання певних протоколів	+	можливе	дуже бідна документація
Sulley	Python	Будь-яка, яка підтримує мову Python	Python	блокове подання	+	гнучке	докладна документація [2]

Peach є міжплатформним ІСФ з відкритим початковим кодом, написаним на мові Python. У порівнянні з іншими доступними фреймворками фаззінгу Peach, 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, с. 381].

Генератори відповідають за генерацію даних — від простих рядків до багатошарових двійкових повідомлень. Для спрощення процесу генерації складних типів даних генератори можуть об’єднуватися у ланцюжки. Перетворювачі змінюють дані та можуть об’єднуватися у ланцюжок та 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, с. 382].

Виходячи з теоретичних засад Peach є досить ефективним фреймворком.

GPF був розроблений для використання на платформі Unix і має відкритий початковий код. Основна перевага цього ІСФ полягає у мінімальних витратах, необхідних для початку розробки та запуску фаззеру. 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, с. 387]. 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 інтегрованих середовищ фаззінгу 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 евристиками, побудовою певних протоколів та реалізує інші функції. Зокрема, він підтримує NNTP, SMTP, IMAP та інші протоколи [3, с. 148; 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 ІСФ 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 контрольних прикладів призвела до появи помилок.

Висновки. Таким чином, було запропоновано для порівняльного аналізу ІСФ використати наступні їх основні характеристики: мова реалізаці; 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 швидкого написання фаззерів або розробки нової фаззінгової системи виявлення уразливостей.

Список літератури

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.