УДК 004.056.55

Дослідження вразливості переповнення буфера у комп'ютерних системах

Хох В.Д., аспірант,
Мелешко Є.В., к.т.н., доцент
Центральноукраїнський національний технічний університет, м. Кропивницький

Розповсюдження інформаційних технологій у сучасному світі значно збільшує об'єм чутливої інформації, що зберігається або циркулює мережами. Разом із зростанням об'ємів чутливої інформації постійно зростають і вимоги до систем захисту цієї інформації. Великі успіхи у галузі інформаційної безпеки зроблені зі сторони криптографічних засобів. Зараз приблизно 11% українського трафіку захищено за допомогою SSL [1], і це число зростає. Але попри все, деякі вразливості залишаються незмінними, однією з таких є переповнення буфера.

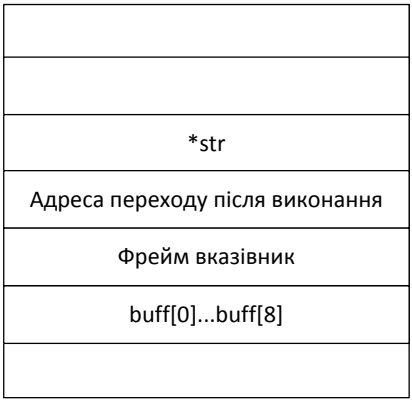
Переповнення буфера стеку є дуже примітивною проблемою, тим не менш вона є однією з найбільш поширених і найчастіше використовуваних. Варто зауважити, що ця проблема характерна для коду програми, який безпосередньо виконується на процесорі комп'ютера.

Розглянемо приклад вразливої програми та її стеку (рис. 1):

```
void overflowedfunc(char *str)
{
	char buff[9];
	strcpy(buff, str);
}

Int main(int argc, char **argv)
{
	char bbuff[10];
	overflowedfunc(bbuff);
	return 1;
}
```

Вразлива програма



Стек виклику overflowedfunc

Рис. 1 – Приклад вразливої програми та стеку виклику вразливої функції

В даному прикладі функція overflowedfunc є вразливою. Проблема полягає у використанні функції strcpy, яка не перевіряє розмір буфера і копіює його безпосередньо. Таким чином, в даній програмі після виклику

функції overflowedfunc буде перезаписана пам’ять, яка знаходиться зверху від блока з buff. Блок з адресою переходу містить адресу пам’яті, на яку буде виконано перехід, і з якої буде продовжено виконання інструкцій.

Використання вразливості переповнення буфера полягає у спробі переписати адресу переходу таким чином, щоб вона вказувала на той набір інструкцій, який цікавить спеціаліста. Для використання цієї вразливості, окрім відомостей про вразливі механізми програми, необхідно розмістити безпосередньо набір інструкцій у пам’яті комп’ютера-жертви. Наприклад, якщо вразлива функція виконує обробку певних даних з файлу – тоді ці інструкції має сенс розмістити у самому файлі, що виконає переповнення. Ще однією з проблем, з якою стикнеться спеціаліст – це визначення адреси, з якої починається його набір інструкцій, на практиці це вирішується за допомогою побудови так званих NOP sled [2] – це щось на зразок посадкової смуги для вказівника переходу. Ідея полягає у записі іноді тисяч або сотень інструкцій NOP (no-op) перед набором інструкцій, які необхідно виконати. NOP`и вказують процесору нічого не робити (no operation – англ. немає інструкції). Використовуючи такий прийом, спеціалісту необхідно знати лише приблизне місце розташування свого набору інструкцій.

Вразливість досить примітивна, і це робить її досить поширеною, оскільки перевірка розмірів буферів часто буває проігнорованою у зв’язку з уявленням про те, що розроблювана програма буде працювати у штатному режимі і функції, що використовують/викликають вразливі функції просто не можуть передати данні більших розмірів. Іноді програмісти вказують просто великий розмір буферу з розрахунку на те, що нікому в голову не прийде спробувати передати їх програмі більшу кількість даних, або, наприклад, обмежують розміри полів вводу. Також варто зауважити, що концепція віртуальної пам’яті, що поширена в сучасних операційних системах, дозволяє проводити аналіз програмного забезпечення на наявність такого роду вразливостей у досить детермінованому просторі, що робить можливим цей процес частково автоматизувати. До того ж, концепція віртуальної пам’яті дозволяє використовувати вразливість на різних машинах для однакових вразливих програм.

Список літератури

1. Статистика українського інтернету [Електронний ресурс] // ukralio. – 2017. – Режим доступу до ресурсу: https://www.ukralio.com/statistika-ukrainskogo-internetu

2. How security flaws work: The buffer overflow [Електронний ресурс] // arstechnica. – 2015. – Режим доступу до ресурсу: https://arstechnica.com/security/2015/08/how-security-flaws-work-the-buffer-overflow/