Симеон Кръстев
ФН: 44672

Android Stagefright Exploit

Курсов проект

Основи на Сигурното Уеб Програмиране

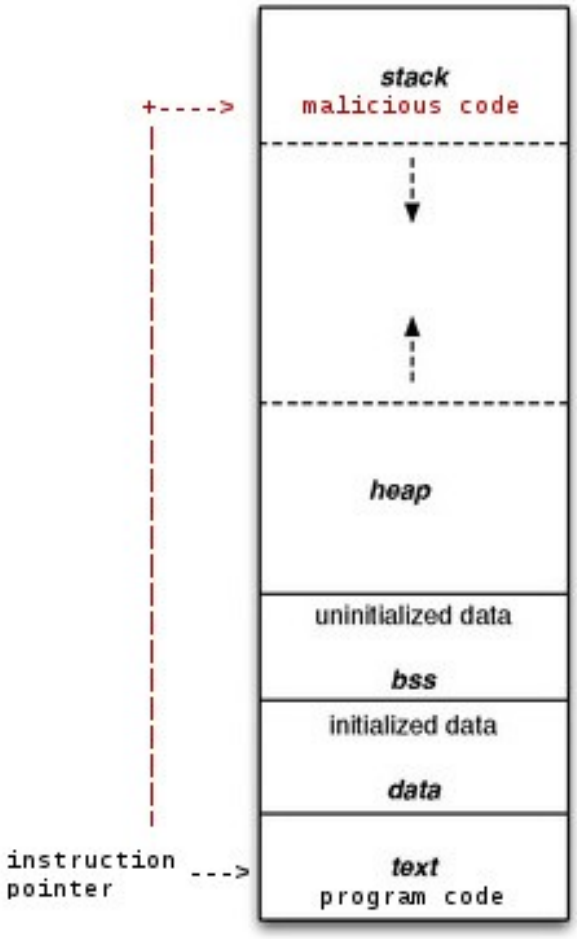
1. Въведение

В ерата на Уеб технологиите основният фокус на сигурността е към информацията протичаща по мрежата – криптиране на съобщения, поверителност на сесиите, защита от вмъкване на код в уеб форми. Все пак управлението на паметта на най-ниско ниво продължава да отваря уязвимости, при това включително и в мобилната сфера. През 2015 от Zimperium откриват пропуски в дълбините на Android, които отварят множество възможни атаки – бъга Stagefright. Това прави милиони, по това време, мобилни устройства потенциални жертви на нежелан достъп до данните и функционалности, като камерата, аудио потоците, блутут и други периферии. Надолу ще бъдат разгледани основната идея за атака през паметта, пропуските в Android, възможностите за експлоатация и защитите.

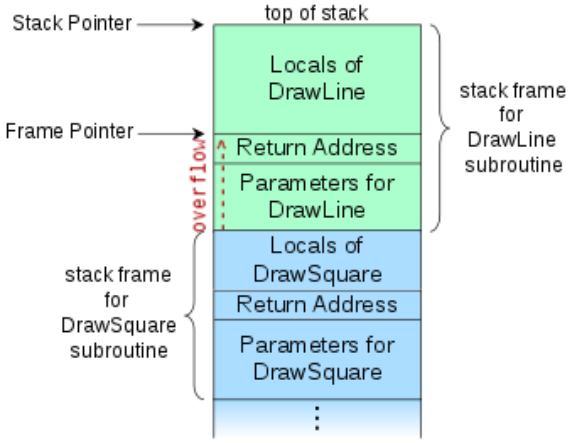
2. Атаки на паметта

Принципът на действие на атаките през паметта разчита на това да бъде променен първоначалният ход на процесора. Това може да стане като на потребителският вход на програмата бъдат подадени некоректни данни, които могат да блокират процеса, да породят нежелан резултат, или дори да съдържат готов за изпълнение код.

За да се обяснят методите за атака, първо трябва да бъдат поставени някои понятия при x86 архитектурата:



- Program code – в началото на адресното пространство се записват инструкциите към процесора.
- Heap – памет заделяна динамично по време на изпълнение на програмата.
- Stack – статично заделяна памет съдържаща локалните и служебни данни за функциите.
- Instruction pointer – указател сочещ текущо изпълняваната инструкция в програмния код.
- Return address – “реда” в програмния код към който да премине instruction pointer-а при преключване на функция.



Възможности за атаки над този модел включват:

- Integer overflow – Подават се параметри такива, че да накарат някоя променлива да надхвърли максималната си стойност. Тогава стойността се пренасища и остатъка се добавя към минималната. Това дава възможност да бъдат прескочени конкретни проверки.
- Buffer overflow – Записвайки променливи в паметта, както stack, така и heap, x86 архитектурата допуска обем данни по-голям от заделената памет. Това позволява нарушаване на паметта на програмата и непредвидено поведение.

Stack overflow attack

Най-простата схема за ефективна атака е тази над стека. Следния фрагмент илюстрира уязвима програма. Тя заделя 50 байта за локална променлива в стека и копира вътре параметър от входа.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(int argc, char** argv)
5  {
6   char buffer[50];
7   strcpy(buffer, argv[1]);
8   printf("%s\n",buffer);
9   return 0;
10 }
```

Уязвимостта тук идва от това, че потребителят може да въведе произволно голям вход. Процесът ще преключи със Segmentation fault. Само това би породило DoS (Denial of Service).

По-сериозният проблем е възможността на потребителя да подготви точно такъв вход, че да презапише някой от програмните регистри – най-често return address-а за текущата функциця с подбрана от него стойност. В останалата част от входа може да запише предварително подготвени инструкции за процесора и да насочи instruction pointer-а именно към тях.

Функции като **strcpy**, **memcpy** не проверяват границите и размерите на параметрите си и така правят възможна най-простата атака върху паметта. Така само подавайки точни параметри може да бъде накаран процесора да изпълнява външен за програмата, дори машината код.

3. Stagefright

Android Architecture

Основната опасност на Stagefright бъга се крие именно в архитектурата и начина на работа на Android. Ето някои основни момента:

- **Isolation / Users and Groups** – Функционалностите на устройството работят в отделни процеси без достъп помежду им. За комуникация се използват т.нар Intents – предоставен от Android механизъм за размяна на съобщения. Правата за достъпсе

управляват като на процесите се дава User, участващ в различни групи. Различните групи дават различни права на контрол – **root**, **system**, **network**, **camera**, **radio**, etc…

- **Startup procedure**
  - Bootloader – зарежда firmware-а за отделните периферии и операционната система
  - Kernel – инициализира паметта и вградените функционалности
  - User space – зареждат се потребителските програми
- **init.rc** – Системен файл който се изпълнява при зареждане на linux kernel-а. В него е описана процедурата по стартиране на операицонната система. Освен това в него различните OEM (производители) описват service-ите оправляващи техните периферии, както и правата им за достъп. init.rc отговаря за това да поддържа service-ите винаги на линия.

Attack Surface

Една от най-популярните уязвимости през паметта в Android, е открита в компонент от ниско ниво на C++, използван за разчитане на различни медия файлове – mp3, mp4, wav, mov, avi, etc…

- **MediaServer** – системен service за четене и управление на мултимедия. Стартира се в init.rc и се извиква от множество приложения – браузъри, галерии, плеъри, чатове… Получава значително количество права – винаги **audio**, **camera**, често случаи **system**, **graphics**, **net**, а в някои случаи дори и **radio**.

```
service media /system/bin/mediaserver
    class main
    user media
    group audio camera net_bt net_bt_admin dramrpc mediadrm
```

- **libstagefright** – извиква се от MediaServer за всеки прочетен файл. Използва се за извличане както на самият аудио/видео stream, така и на мета-данни за файла. Разбива файла на малки парчета – chunks, като всяко отговаря за различна информация.
- **parseChunk** – функцията която отваря уязвимостта. Извиква се при всеки опит файла да бъде прочетен и да бъде извлечено каквото и да е от него.

```
/* allocate enough memory for both the old buffer and the new buffer */
uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
if (buffer == NULL) {
    return ERROR_MALFORMED;
}

/* if there was any previous timed-text data */
if (size > 0) {
    /* copy the data to the beginning of the buffer */
    memcpy(buffer, data, size);
}
```

При четене на отделен chunk не се прави проверка за размера му, преди да бъде копиран в локалната памет. Това дава възможност да бъде подаден “фалшив” видео файл, който съдържа в себе си buffer overflow атака.

Attack Vectors

Няколко сценария за експлоатиране на описаната уязвимост:

- **Вграден видео файл в уеб страница** – Когато браузъра срещне **<video>**, или **<media>** тагове, ще се опита да ги свали локално и да ги прочете за изобразяване. Така операционната система ще се обърне към MediaServer-a, а той от своя страна към libstagefright. Така само с отваряне на уеб страница може да бъде изпълнен външен за мобилното устройство код.
- **Auto download policy** – много проложения, включително Hangouts, Facebook и вграденият Messanger по подразбиране свалят всички получени файлове. Още при свалянето Android ще се опита да ги каталогизира, за по-лесно управление после. Дори без да се отвори файла за разглеждане, само за да бъдат прочетени метаданните отново ще се направи извикване до MediaServer и libstagefright.
- **USB / SD Card** – при поставяне на външен storage Android автоматично сканира всички файлове на него и ги каталогизира. Отново MediaServer, отново libstagefright.
- **MMS** – най-опасният от всички сценарии. При гореизброените все пак на първо място потребителя сам с действия стига до файла. Android обаче автоматично сваля и сканира също получените MMS съобщения. Така фалшивият код може да бъде изпълнен без никаква интеракция от потребителя.

Възможности

Възможностите за експлоатация варират между различните версии и модели устройства, в зависимост от това какви права са дадени на mediaserver процеса.

- Стартиране на процес наследник със същите права като mediaserver-a. Миньор за крипто-алгоритми, бот за DDoS, или просто да се блокира устройството
- Директно четене на файлове от файловата система – снимки, документи, пароли, private-keys
- Подслушване на audio-stream-a и насочването му към външен хост.
- Управление на камерата
- При моделите даващи **radio** права на MediaServer-a, това позволява дори правене на обаждания, изпращане на съобщения, ефективно размножаване на атаката…

4. Защити

Начините за пресичане на такива атаки варират от недопускането на нарушения в паметта на ниско ниво, до правилни политики и решения на високо.

- **ASLR (Address Space Layout Randomization)** – Техника използвана от операционните системи за случайно разпределяне на променливите в паметта. Праи много по-трудно да се предвиди къде ще бъдат адресите на различните регистри.
- **NX/DEP (Non-Execute / Data Execute Prevention)** – Флагове отбелязващи отделни сегменти в паметта, които процесора отказва да изпълни. Това забранява на instruction pointer-а да прескочи в data сектора и да
- **Canaries** – Защитни битове поставяни точно преди return address-а. Когато бъдаат презаписани това е индикация за buffer overflow. Програмата може да терминира и да инвалидира данните.
- **Type safety / bounds check** – Използване на праилни типове и подробни проверки за размери и допустими стойности.

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char** argv)
{
    if (strlen(argv[1]) > 50) {
        printf("Malicious code detected!\n");
        return 1;
    }
    char buffer[50];
    strcpy(buffer, argv[1]);
    printf("%s\n",buffer);
    return 0;
}
```

- **Защити на високо ниво** – Адекватни правила при приемане на файлове. Подходящо раздадени права и привилегии.

Заключение

В последните години основната част от IT сферата вече е в уеб комуникациите и езиците от високо ниво. Те не дават достъп за пряко управление на паметта и хода на процесора, но въпреки това всяко приложение използва модули и компоненти от ниско ниво. Винаги ще имаме нужда от прецизни изчисления и оптимални алгоритми. Нагоре бяха показани принципите по които такива компоненти могат да бъдат компрометирани и възможните последици от това. Сега когато обмяната на данни между всякакви устройства от всякакви източници е по-интензивна от всякога, внимателната проверка на данните дори на най-ниско ниво остава ключов фактор за дигиталната сигурност.