

Anti-cheating Engine for Android

Error rendering macro 'html'

Notify your Confluence administrator that "Bob Swift Atlassian Apps - HTML" requires a valid license. Reason: VERSION_MISMATCH

Excuse the ads! We need some help to keep our site up.


Error rendering macro 'html'

Notify your Confluence administrator that "Bob Swift Atlassian Apps - HTML" requires a valid license. Reason: VERSION_MISMATCH

List

- [Anti-cheating Engine for Android](#)
 - [Check modification of binaries](#)
 - [Check debug](#)
 - [Check Speed hacking](#)
 - [Check Virtual Machine](#)
 - [Related site](#)

Anti-cheating Engine for Android

-  해당 내용은 2016년에 분석하고 개발한 Anti-cheating Engine for Android에 대한 설명입니다.
 - <https://github.com/Lazenc/Lazenc-S>
- 여기에서 설명하는 내용들은 아주 기초적인 기술들에 대한 설명입니다.
- 아래 기능들은 최신 Cheat tool에서 우회 될 수 있습니다.

Check modification of binaries

- 아래와 같이 배포된 파일의 변조를 확인하기 위한 간단한 기능입니다.
 - 파일 변조를 확인할 파일의 Hash 정보를 추출해서 서버에서 배포본의 Hash 정보와 동일인지 확인합니다.
- 해당 코드를 구현 할 때 주의 할 부분들이 있습니다.
 - 변조 유무를 확인 할 파일의 경로는 암호화되어 서버 또는 파일에 전달,보관되어 있어야 합니다.
 - 변조 유무를 확인 할 파일의 경로를 코드 또는 파일에 보관 할 경우 Reverse engineering을 통해 해당 경로 또는 파일명을 변경하여 우회할 수 있습니다.
 - Hash 알고리즘으로 CRC와 같이 취약성이 존재하는 알고리즘을 사용해서는 안됩니다.

CheckBinaryModification.h

```
bool ModFindHash(char *hash) {  
  
    long lSize;  
    size_t result;  
    int result_val;  
  
    char *buffer;  
    char patternFile[MAX_PATH] = "/mnt/sdcard/pattern";  
  
    FILE *fp = fopen(patternFile, "rb");  
  
    if (fp) {  
        fseek(fp, 0, SEEK_END);  
        lSize = ftell(fp);  
    }
```

```

rewind(fp);

buffer = (char*) malloc(sizeof(char) * lSize);
if (buffer == NULL) {
    fputs("Memory error", stderr);
    exit(2);
}

result = fread(buffer, 1, lSize, fp);
if (result != lSize) {
    fputs("Reading error", stderr);
    exit(3);
}

if (result > 0) {
    if(strstr(buffer,hash)){
        result_val = true;
    }else{
        result_val = false;
    }
}
fclose(fp);
free(buffer);
} else {
    return false;
}
return result_val;
}

void CheckModBinary(){
    pid_t pid= getpid();

    char sha256[65] = "";

    char packageName[MAX_PATH];
    char apkFilePath[MAX_PATH];
    char libFilePath[MAX_PATH];
    char tmpLibFilePath[MAX_PATH];

    int zipcount = 0;
    unsigned int crc;

    DbgPrint("Modification Check");

    getCmdline(pid,packageName,sizeof(packageName));
    getApkFilePath(packageName,&apkFilePath);

    sprintf(libFilePath, "/data/data/%s/lib", packageName);

    while(1)
    {
        fileToSha256(&sha256,apkFilePath);

        if(ModFindHash(sha256) == false)
        {
            DbgPrint("File Path : %s, SHA256 : %s - Modification",apkFilePath,sha256);
            DbgPrint("Modification.");
            sleep(10);
            exit(0);
        }

        DIR *dirInfo;
        struct dirent *dirEntry;

        dirInfo = opendir(libFilePath);
        if (NULL != dirInfo) {
            while (dirEntry = readdir(dirInfo))
            {
                sprintf(tmpLibFilePath, "/data/data/%s/lib/%s", packageName,dirEntry->d_name);

                fileToSha256(&sha256,tmpLibFilePath);
            }
        }
    }
}

```

```

        if (ModFindHash(sha256) == false) {
            DbgPrint(".SO File Hash Check - Modification.");
            sleep(10);
            exit(0);
        }
    }
    closedir(dirInfo);
}
sleep(1000);
}
}

```

Check debug

- 아래와 같이 두가지 방식으로 디버깅 여부를 확인할 수 있습니다.
 - 첫번째는 해당 프로세스의 `/proc/pid/cmdline` 파일의 내용을 확인하는 것 입니다.
 - 프로그램이 GDB에 의해 실행 될 경우 PPID의 프로세스는 gdb이기 때문에 `cmdline` 파일을 이용해 gdb를 탐지 할 수 있습니다.

bool isCheckCmdline()

```

bool isCheckCmdline() {
    char filePath[32], fileRead[128];
    FILE* file;

    snprintf(filePath, 24, "/proc/%d/cmdline", getpid());
    file = fopen(filePath, "r");

    fgets(fileRead, 128, file);
    fclose(file);

    if(!strcmp(fileRead, "gdb")) {
        DbgPrint("Debugger(gdb) detected\n");
        return true;
    }
    DbgPrint("Clear(Debug)\n");
    return false;
}

```

- 두번째는 해당 프로세스의 `/proc/pid/status` 파일의 내용을 확인하는 것 입니다.
- `status` 파일에는 해당 프로세스의 상태, 스레드 정보, 메모리 정보, 등 많은 정보들이 저장되어 있습니다.
- 여기에서 Debug 여부를 확인하기 위해 `TracerPid`의 값을 이용 할 수 있습니다.
 - 해당 프로세스가 다른 프로세스에 의해 추적 되고 있다면 추적하고 있는 프로세스의 PID가 저장됩니다.
 - 추적이 되고 있지 않으면 '0' 이 저장됩니다.

bool isCheckTracerPid()

```
bool isCheckTracerPid() {
    int TPid;
    char buf[512];

    const char *str = "TracerPid:";
    size_t strSize = strlen(str);

    FILE* file = fopen("/proc/self/status", "r");

    while (fgets(buf, 512, file)) {
        if (!strncmp(buf, str, strSize)) {
            sscanf(buf, "TracerPid: %d", &TPid);
            if (TPid != 0) {
                DbgPrint("Debugger detected\n");
                return true;
            }
        }
    }
    fclose(file);
    DbgPrint("Clear(Debug)\n");
    return false;
}
```

Check Speed hacking

- Memory cheat에서 제공하는 Speed hack은 시스템에서 제공하는 시간 관련 함수(gettimeofday, ...)를 후킹하여 값을 조절함으로써 게임의 속도를 조절할 수 있습니다.
- 다음과 같은 방법으로 Speed hack을 확인 할 수 있습니다.
 - 시스템에서 제공하는 시간 관련 함수로 부터 기준 값(start time)이 될 시간 값을 받아옵니다.
 - 그리고 sleep() 함수를 이용해 원하는 시간 만큼 프로세스를 정지 시킵니다.
 - sleep() 함수가 종료 된 후에 다시 시간 관련 함수로 부터 시간 값(end time)을 받아옵니다.
 - 두 시간 값을 연산(end time - start time)을 통해 sleep() 함수에서 소비된 시간을 구 할 수 있습니다.
 - 이렇게 연산된 값이 sleep() 함수에 전달한 시간 값과 비슷하지 확인함으로써 Speed hack 여부를 확인 할 수 있습니다.
 - Ex)
 - gettimeofday() 함수를 이용할 경우 sleep(100) 일 경우 연산된 값은 101000 보다 작거나 10000 보다 커야 합니다.
 - clock_gettime() 함수를 이용할 경우 sleep(100) 일 경우 연산된 값은 100001000 보다 작거나 100000000 보다 커야 합니다.
- 최근 Memory cheat에서는 시간과 관련된 대부분의 함수를 후킹하여 값을 변조하고 있기 때문에 해당 코드가 효과를 보기 어려울 수 있습니다.

CheckSpeedHack.h

```
int getTime1()
{
    struct timeval tv;
    struct timezone tz;
    gettimeofday (&tv, &tz);
    return (tv.tv_sec*1000 + tv.tv_usec/1000);
}

long getTime2()
{
    struct timespec now;
    clock_gettime(CLOCK_MONOTONIC,&now);
    return now.tv_sec*1000000 + now.tv_nsec/1000;
}

void CheckSpeedHack(){
    int start = 0, end = 0, time_data = 0;
    int start2 = 0, end2 = 0, time_data2 = 0;
    int cmp=1;

    while(cmp){
        start = getTime1();
        start2 = getTime2();

        sleep(100);
        end = getTime1();
        end2 = getTime2();

        time_data = end - start;
        time_data2 = end2 - start2;
        DbgPrint("time_data : %d, end : %d , start : %d",time_data,end,start);
        DbgPrint("time_data2 : %d, end : %d , start : %d",time_data2,end2,start2);
        if(time_data > 101000 || time_data < 10000){
            cmp = 0;
            DbgPrint("SpeedHackDetect");
            sleep(10);
            exit(0);
        }
        if(time_data2 > 100001000 || time_data2 < 100000000){
            cmp = 0;
            DbgPrint("SpeedHackDetect");
            sleep(10);
            exit(0);
        }
    }
}
```

Check Virtual Machine

- 다음과 같은 방법으로 Virtual Machine을 확인 할 수 있습니다.
 - "/system/build.prop" 파일에 저장된 정보를 이용 할 수 있습니다.
 - "/system/build.prop" 파일에 Android 시스템에 관한 다양한 정보들이 저장되어 있습니다.
 - 특히 Virtual Machine의 경우 아래 항목들에 Virtual Machine의 이름을 설정하고 있으며, 이러한 정보를 이용해 Virtual Machine을 탐지 할 수 있습니다.
 - "ro.product.manufacturer"
 - "ro.product.model"
 - "ro.product.name"
 - "ro.product.device"
 - 하지만 해당 항목의 값을 변경하는것으로 탐지를 우회 할 수 있습니다.
- 해당 방법 외에도 다양한 방법들이 있습니다.
 - 각 Virtual Machine에서 설치 또는 실행 중인 프로세스의 여부 확인
 - 시스템 구조를 분석하여 Virtual Machine에서 추가한 부분을 확인
 - 등등

CheckVirtualMachine.h

```
void virtualMachine(char *filePath, char *searchStr, char *findStr){
    FILE *fp;
    char str[81];
    fp = fopen(filePath, "r");

    while(!feof(fp))
    {
        fgets(str, 80, fp);
        if(strstr(str, searchStr) != NULL){
            str[strlen(str)-1] = ',';
            strcat(findStr, str);
        }
    }
    fclose(fp);
}

bool IsBlackDevice(char *deviceName)
{
    if(strstr(deviceName, "BlueStacks") != NULL)
    {
        DbgPrint(" BlueStacks : %s,%p", deviceName, deviceName);
        return false;
    }

    if(strstr(deviceName, "Genymotion") != NULL)
    {
        DbgPrint(" Genymotion : %s,%p", deviceName, deviceName);
        return false;
    }
    return true;
}

void CheckVirtualMachine(){
    char findSearch[300];

    virtualMachine("/system/build.prop", "ro.product.manufacturer", &findSearch);
    virtualMachine("/system/build.prop", "ro.product.model", &findSearch);
    virtualMachine("/system/build.prop", "ro.product.name", &findSearch);
    virtualMachine("/system/build.prop", "ro.product.device", &findSearch);

    if(IsBlackDevice(findSearch)) {
        DbgPrint(" BlueStacks : Not Detect");
    }else{
        DbgPrint(" BlueStacks : Detect");
        sleep(10);
        exit(0);
    }
}
```

Related site

- <https://github.com/Lazenca/Lazenca-S>

Error rendering macro 'html'

Notify your Confluence administrator that "Bob Swift Atlassian Apps - HTML" requires a valid license. Reason:

VERSION_MISMATCH

