Course material is not allowed during the exam.
Try to keep your answers **precise and short**. You will **not get extra points** by giving very long answers or by writing down what you know instead of what is asked.
Just answer the question that is asked.
Take 10 seconds to think about what you are going to write before writing it.

---

**Homeland Security: Disable UPnP as tens of millions at risk**

**Summary:** The US government is warning to disable a common networking feature after bugs left tens of millions of hardware devices vulnerable to attacks by hackers and malware.

By Zack Whittaker for Zero Day | January 29, 2013 -- 21:03 GMT (22:03 CET)
Follow @zackwhittaker

The US Department of Homeland Security is next in line to warn of a serious threat to networking devices, such as scanners, printers, computers, and routers.

[...]

1. Race conditions.
   a) Explain what is a race condition in one sentence.

   b) Give an example of an attack using a race condition to attack a system.

   c) Cite 2 methods to prevent, or mitigate, race conditions.

2. Terminology.
   a) What is a threat?

   b) What is a vulnerability?

   c) What is an exploit?
3. Check the boxes to say whether the assertions below are valid or not (good answer 1, bad answer -1, result for this question between 0 and 5):

<table>
<thead>
<tr>
<th>Assertion</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>An integer overflow can lead to a stack-based buffer overflow.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Oriented Programming defeats non executable stacks (W xor X).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Oriented Programming defeats ASLR.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory leaks helps to bypass ASLR.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory leaks helps to bypass stack canaries.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String format bugs are still exploitable today, despite the presence of mitigation techniques (PAX, ASLR).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. In the 70's Saltzer and Schroeder defined 8 security principles, among them “Least privilege” and “Separation of privilege”. Explain the difference between the two.

5. What is the attack surface of a system?

6. Explain under which circumstances a null pointer dereference bug can be exploited.

7. What is a Dynamic root of trust?
8. CAPTCHAs are often used by online services to prevent automated access. Describe one class of attacks against graphical CAPTCHAs that is highly efficient.

9. The first page of the exam contains a screen-shot of a recent news about vulnerabilities in Universal Plug And Play (UPnP) as used in many devices and computers. While there is some sensationalism in there, the issue is actually severe. A recent report from HD Moore (“UPnP unplug and pray?”) discloses several facts about a reference implementation of UPNP. Here are a few excerpts:

“Universal Plug and Play (UPnP) is a protocol standard that allows easy communication between computers and network-enabled devices. This protocol is enabled by default on millions of systems, including routers, printers, media servers, IP cameras, smart TVs, home automation systems, and network storage servers. UPnP support is enabled by default on Microsoft Windows, Mac OS X, and many distributions of Linux.”

The UPnP protocol suffers from a number of basic security problems […]. Authentication is rarely implemented by device manufacturers, privileged capabilities are often exposed to untrusted networks, and common programming flaws plague common UPnP software implementations. These issues are endemic across UPnP-enabled applications and network devices.

The statistics in this paper were derived from five and a half months of active scanning. UpnP discovery requests were sent to every routable IPv4 address approximately once a week from June 1 to November 17, 2012. This process identified over 81 million unique IP addresses that responded to a standard UPnP discovery request. […]

This paper quantifies the exposure of UPnP-enabled systems to the internet at large, classifies these systems by vendor, identifies specific products, and describes a number of new vulnerabilities that were identified in common UPnP implementations. Over 1,500 vendors and 6,900 products were identified that are vulnerable to least one of the security flaws outlined in this paper. Over 23 million systems were vulnerable to a single remote code execution flaw that was discovered during the course of this research. […]

1 UDP packet is all it takes to exploit any of the 8 newly-discovered libupnp vulnerabilities.”

a) Given the above news what should be the steps to take to protect oneself against such attacks?
b) What is the big risk, knowing that “1 UDP packet is enough to exploit the flaw” and that there are millions of devices vulnerables ?

c) Below is a piece of code, the complete copy of one single function, in a popular implementation of UPNP. This function is vulnerable to several attacks. Mark each vulnerabilities you can find together with a very brief explanation/description of the vulnerability, write your answer directly on the listing. There are 7, and most of them are basic buffer overflows ! (If needed, check the man pages sample material at the end of the page).

d) Explain, very briefly, how you would exploit one of those bugs? Provide an example of a string to exploit one vulnerability (of course exact offsets and the shell code are not needed, a sketch of the string is fine).
int unique_service_name( IN char *cmd,
               IN SsdpEvent *Evt )
{
    char *TempPtr,
    TempBuf[COMMAND_LEN],
    *Ptr,
    *ptr1,
    *ptr2,
    *ptr3;
    int CommandFound = 0;
    
    if( ( TempPtr = strstr( cmd, "uuid:schemas" ) ) != NULL ) {
        ptr1 = strstr( cmd, "device" );
        if( ptr1 != NULL ) {
            ptr2 = strstr( ptr1 + 1, ":" );
            if( ptr2 != NULL ) {
                ptr3 = strstr( ptr2 + 1, ":" );
                if( ptr3 != NULL ) {
                    sprintf( Evt->UDN, "uuid:%s", ptr3 + 1 );
                } else {
                    return -1;
                }
            } else {
                return -1;
            }
        } else {
            ptr1 = strstr( cmd, ":" );
            if( ptr1 != NULL ) {
                strcpy( Evt->DeviceType, "urn%s", TempBuf );
            } else {
                return -1;
            }
        }
    } else {
        ptr1 = strstr( cmd, "uuid" );
        if( ptr1 != NULL ) {
            //printf("cmd = %s\n",cmd);
            if( ( Ptr = strstr( cmd, ":::" ) ) != NULL ) {
                strncpyc( Evt->UDN, TempPtr, Ptr - TempPtr );
                Evt->UDN[Ptr - TempPtr] = '\0';
                sprintf( Evt->DeviceType, "urn%s", TempBuf );
            } else {
                return -1;
            }
        }
    }
    return 0;
}
if( strstr( cmd, "urn:" ) != NULL
    && strstr( cmd, ":service:" ) != NULL ) {
    if( ( TempPtr = strstr( cmd, "urn" ) ) != NULL ) {
        strcpy( Evt->ServiceType, TempPtr );
        CommandFound = 1;
    }
}

if( strstr( cmd, "urn:" ) != NULL
    && strstr( cmd, ":device:" ) != NULL ) {
    if( ( TempPtr = strstr( cmd, "urn" ) ) != NULL ) {
        strcpy( Evt->DeviceType, TempPtr );
        CommandFound = 1;
    }
}

if( CommandFound == 0 ) {
    return -1;
}
return 0;

Some background:

- This code path is reachable before authentication, with UDP packets, the cmd variable is controolled by the attacker

- Sample material from the man page of the strncpy function:

  char *strncpy(char *dest, const char *src);
  char *strncpy(char *dest, const char *src, unsigned int n);

  DESCRIPTION

  The strncpy() function copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. The strings may not overlap, and the destination string dest must be large enough to receive the copy.

  The strncpy() function is similar, except that at most n bytes of src are copied. Warning: If there is no null byte among the first n bytes of src, the string placed in dest will not be null-terminated.

  If the length of src is less than n, strncpy() pads the remainder of dest with null bytes.

  RETURN VALUE

  The strcpy() and strncpy() functions return a pointer to the destination string dest.

- From the man page of the strstr function:

  char *strstr(const char *haystack, const char *needle);

  DESCRIPTION

  The strstr() function finds the first occurrence of the substring needle in the string haystack. The terminating null bytes ('\0') are not compared.

  RETURN VALUE

  strstr() returns a pointer to the beginning of the substring, or NULL if the substring is not found.