9 From Protocol to PoC; or, Your Cisco blade is booting PoC||GTFO.

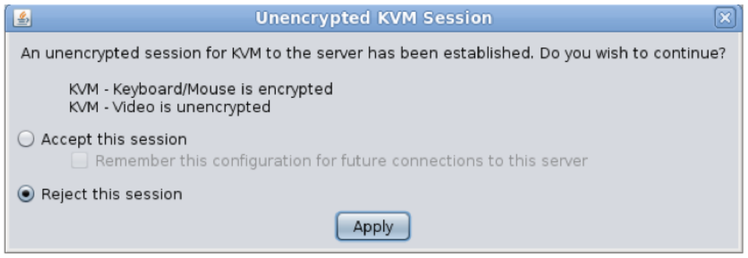
by Mik

We often see products with network protocols intended to be opaque to us. We suspect that we can do interesting things with it, but where do we start?

This article will guide you from an opaque protocol used by Cisco UCS and some Dell servers for KVM and remote virtual media block device functionality, to a PoC that takes advantage of this protocol's bolt-on security. This protocol has been the subject of Bug IDs CSCtr72949 and CSCtr72964, better knows as CVE-2012-4114 and CVE-2012-4115. But then, who among you, when your son hungers for a PoC, would give him a CVE?[14]

So we will walk the road to PoC together, working up to a way to replace the CD/DVD that the administrator is exporting with a more fun virtual ISO image, then take the further step of redirecting the inserted USB key via a more open protocol.

While data centers are near-optimal habitats for computers, spending long hours and late nights there can be quite uncomfortable for humans. To alleviate this problem, most server systems incorporate a BMC management console that provides remote keyboard, mouse, video and virtual media—generally emulating a USB keyboard, mouse, DVD-ROM and removable disk, while also intercepting video output.


Unencrypted KVM Session

An unencrypted session for KVM to the server has been established. Do you wish to continue?

KVM - Keyboard/Mouse is encrypted
KVM - Video is unencrypted

○ Accept this session
☐ Remember this configuration for future connections to this server

● Reject this session

My journey down this road started when a prompt from my Cisco blade popped up. It turned out that while keyboard and mouse sessions could do TLS, the video or virtual media interfaces could not. This told me not only that the most dangerous interface to my systems was insecure, but also the TLS support was bolted-on and thus it wasn't hard to trick a user who didn't read the prompt text carefully.

While much fun could be had intercepting the keyboard and video streams, the importance of securing block device access seemed to be overlooked by those filling in the CVSS score form, so I took it upon myself to prepare a demonstration.

In order to do this, we need to understand the protocol, so let us link arms and take a stroll down PoC lane.

9.1 Framing

Distinguishing the individual frames is an excellent starting point for unraveling an otherwise unknown protocol. Generally speaking, a protocol will send messages in one of the following formats:

Explicit length: Just put the message length at or near the start of the message. Sometimes it's the payload length, other times it includes the length field itself.

Examples of this are the DIAMETER protocol, TLS, and indeed the APCP/AVMP protocols described here.

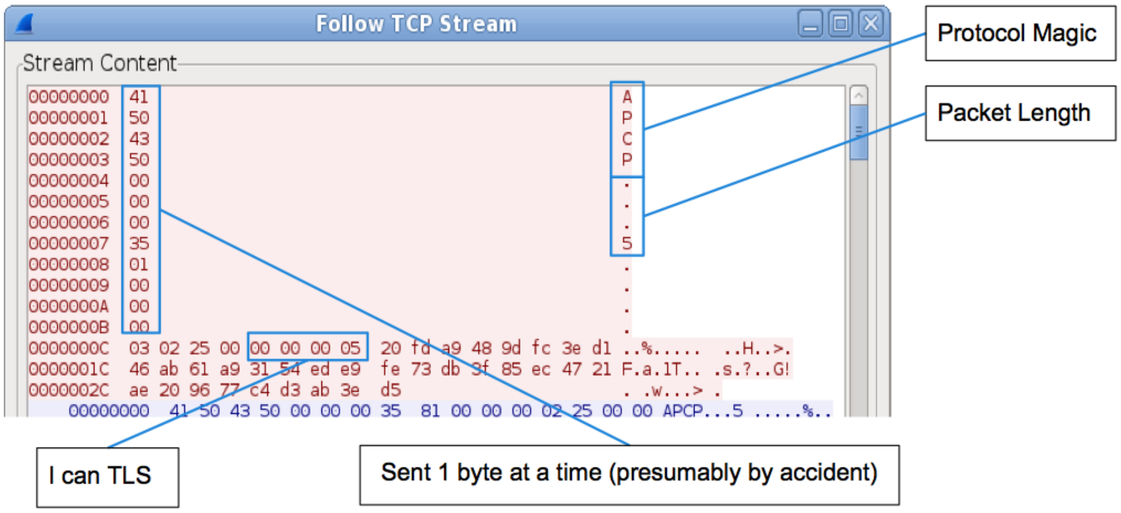
[14] Matthew 7:9

Defer to upper-layer: This is common with UDP-based protocols—simply allow the upper layer to define the frame boundary. It would be foolhardy for a protocol designer to rely on frame boundaries with TCP. Often the sending side will send a complete frame in a segment, offering a vital hint to the reverse engineer.

Delimiter: Classic examples of this are line-oriented protocols such as POP3 and SMTP where the delimiter is CRLF. Other protocols, those originally designed to operate over bitstream transports, refer to their delimiter as "sync bits". The general rule is that the message starts or stops at an easily recognized boundary, and also that they do their damndest to avoid placing the delimiter in the message itself.

Dual-Mode: Even seasoned `vi` users occasionally type code while in command mode or find a rogue `ex` command in a config file. The same can be said for network protocols. HTTP uses CRLF-CRLF as a delimiter to denote the end of the headers, then once the Content-Length header has been parsed the message body length is known. This state transition makes for some awful, buggy implementations, a situation that didn't improve with Chunked encoding.

In our case, the TCP session looks a little something like this.



```
00000000  41                                                A
00000001  50                                                P
00000002  43                                                C
00000003  50                                                P
00000004  00                                                .
00000005  00                                                .
00000006  00                                                .
00000007  35                                                5
00000008  01                                                .
00000009  00                                                .
0000000A  00                                                .
0000000B  00                                                .
0000000C  03 02 25 00 00 00 00 05  20 fd a9 48 9d fc 3e d1 ..%.....  ..H..>.
0000001C  46 ab 61 a9 31 54 ed e9  fe 73 db 3f 85 ec 47 21 F.a.1T.. .s.?..G!
0000002C  ae 20 96 77 c4 d3 ab 3e  d5                       . .w...> .
    00000000  41 50 43 50 00 00 00 35  81 00 00 00 02 25 00 00 APCP...5 .....%..
```

This is extremely lucky, as it seems the application developer accidentally wrote the packet header byte at a time, each having its own segment. This makes it easy to distinguish the header from the body.

As we can see, there's a magic field, "`APCP`", then a big-endian number that happens to match the frame size including the header, then four bytes.

The catch is that there are actually three protocols running on this port: APCP, BEEF, and AVMP, and their respective framing is subtly different.

APCP functions as a control protocol, so we need to decode those frames, even though we're not particularly interested in them.

BEEF is the protocol that the keyboard, video and mouse operate on. We switch to pass-through mode when we see a BEEF packet, or indeed anything we don't recognize, in order to allow it to pass unhindered.

AVMP is the virtual media protocol, which only starts when you click on the virtual media tab. The term "virtual media" may be more familiar if you rephrased it as "remote DVD-ROM and removable disk."

9.2 Message Types

Binary protocols like these generally require that the type of message be in the message header. This is analogous to the request line in HTTP, in that it allows the remote end to route the message to the correct processing routine.

Often enabling logging on the application will simply name the decoded message type for you.[15] There's no need to over-extend yourself decoding particular message types if they don't seem relevant to your PoC, but you should at least note the name and function of messages if you can infer them.

In this case we are dealing with block devices. Block device protocols only have two methods of interest.

```
read(offset, length) -> data[length] | error
write(offset, data[length]) -> ack | error
```

Offset and length are either multiplied by the block size or aligned to the block size. Block devices don't let you write half-blocks—when you write less than a full block to the middle of a file, your filesystem needs to read in the block and write back the modified version.

The read response and write request were easy to spot—simply transfer some data and you'll see it in the frame. The server will send a maximum of sixteen blocks per read response, but will respond in full using multiple messages then send a "Status" message with a code of zero. Error messages are simply "Status" messages with a non-zero code.

Note that in the case of AVMP and NBD (and indeed modern SCSI and ATA protocols) requests are tagged. Each tag is an opaque value on the request, which must be returned with the response. This allows multiple messages to be in-flight at once, which greatly increases the throughput.

Read requests in AVMP also have a third argument, referred to as the Block Factor, which is the maximum number of blocks the application should send back in a single read response. I did not try sending more, mostly because I wished to avoid an unpleasant trip to the data center.

There were other AVMP requests that I had to find and decode. These were the ones that described the drive, and mapped and unmapped a drive (read: inserted or removed a disk).

9.3 TLS

In this age of mistrust, customers are demanding encryption for all of their network protocols. TLS is the standard answer; while it isn't much fun to circumvent TLS, it's generally not much trouble.

If the program talks some cleartext protocol before sending a TLS `ClientHello`, chances are that it is negotiating whether or not to enable TLS over the network. This is, of course, ridiculous, but alas it's a popular idiom for bolted-on cryptography.[16]

In these circumstances, the prudent thing to do would be to tell the client that the server doesn't know what TLS is. My PoC does this with the `--downgrade` option.

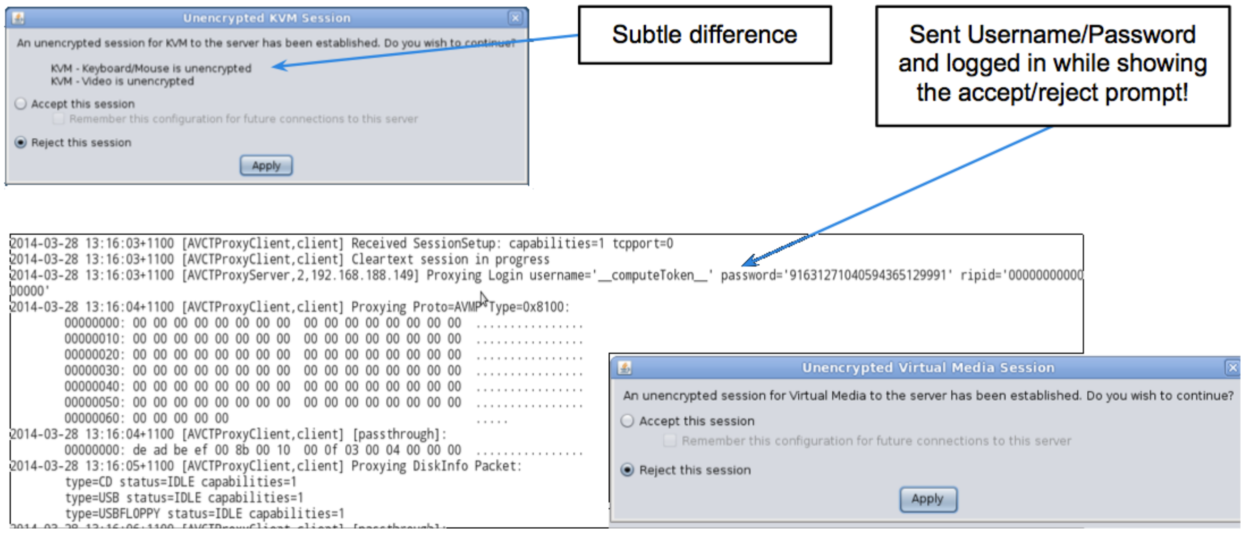
```
Client -> KVM: Session please, I can do TLS
KVM -> Client: Ok, let's TLS
[ TLS negotiation ]

Client -> KVM: Session please, I can do TLS
KVM -> Client: Ok, let's talk plaintext
```

The server often enforces that only TLS connections should be allowed, but since the client is rarely authenticated at the TLS layer, your exploit tool may simply establish a TLS connection to the server while maintaining a cleartext connection to the client.

The effects of connection downgrade are rather subtle. While the connection is now operating in malleable cleartext, the prompt dialog changes only slightly:

[15]"Trace logging" in Java.

[16]Try this with your favorite SMTP, XMPP and IMAP clients—you may be unpleasantly surprised.


Unencrypted KVM Session
Subtle difference
Sent Username/Password and logged in while showing the accept/reject prompt!
Unencrypted Virtual Media Session
Apply

An unencrypted session for KVM to the server has been established. Do you wish to continue?

KVM - Keyboard/Mouse is unencrypted
KVM - Video is unencrypted

Accept this session
Remember this configuration for future connections to this server
[✓ Reject this session]

```
2014-03-28 13:16:03+1100 [AVCTProxyClient,client] Received SessionSetup: capabilities=1 tcpport=0
2014-03-28 13:16:03+1100 [AVCTProxyClient,client] Cleartext session in progress
2014-03-28 13:16:03+1100 [AVCTProxyServer,2,192.168.188.149] Proxying Login username='__computeToken__' password='91631271040594365129991' ripid='0000000000
00000'
2014-03-28 13:16:04+1100 [AVCTProxyClient,client] Proxying Proto=AVMP Type=0x8100:
        00000000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
        00000010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
        00000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
        00000030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
        00000040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
        00000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
        00000060: 00 00 00 00 00                                    .....
2014-03-28 13:16:04+1100 [AVCTProxyClient,client] [passthrough]:
        00000000: de ad be ef 00 8b 00 10  00 0f 03 00 04 00 00 00  ................
2014-03-28 13:16:05+1100 [AVCTProxyClient,client] Proxying DiskInfo Packet:
        type=CD status=IDLE capabilities=1
        type=USB status=IDLE capabilities=1
        type=USBFLOPPY status=IDLE capabilities=1
```

An unencrypted session for Virtual Media to the server has been established. Do you wish to continue?

Accept this session
Remember this configuration for future connections to this server
[✓ Reject this session]

It should be noted that with the virtual media component on the Cisco blades it actually sends the cleartext password in the background before you mindlessly click "Accept".[17]

If the client seems to only wish to talk TLS, an alternative approach may be used. You simply start up a TLS server and accept the client connection. You may then establish a TLS client connection to the server, and forward the data between them. This is commonly called a Man-in-The-Middle attack, but in this modern age it's generally machines rather than men or women who perform such work.

Astute readers will note that this will annoy the certificate validation routine in the client application. In reality, this is rarely the case.[18] If such a validation routine even exists, it can be bypassed with an Accept/Reject dialog which displays some textual information that you can easily duplicate in your own self-signed certificate.

For a particularly ironic example of this, look at the code in the supplied PoC. The two useful options work together with some way of passing the IP traffic to the Machine-in-the-Middle, which runs the client.

```
--servercert SERVERCERT
                    File containing the server certificate for MitM
--serverkey SERVERKEY
                    File containing the server private key for MitM
```

Your friendly neighborhood `iptables` can take care of the redirection.

```
iptables -A PREROUTING -d [target IP] -p tcp --dport 2068 -j REDIRECT --to-ports 2068
```

9.4 Clients and Servers

It is interesting to note that in SCSI there are no clients and servers. Instead, there are Initiators and Targets. This applies to many protocols which two distinct roles, both providing services to each other. The classic example is that a web browser provides more valuable information to the web server than vice versa, yet the reason it's considered the client is that it initiates the connection.

When intercepting network connections, you should consider what services both ends of the connection provide you.

In our example, which intercepts Virtual Media connections between a Java application and BMC, the BMC provides the service of connecting CD-ROMs and removable media to it. While generally this involves

[17]This is still an improvement over other vendors, which do not display any prompt and simply talk in the clear. At least one has devoted man-hours to fixing this since trying out my PoC.

[18]*If you don't believe us, neighbor, there's an academic paper about that, "The most dangerous code in the world: validating SSL certificates in non-browser software", by Georgiev et al. —PML*

a server administrator wasting hours waiting for an operating system to install, we might choose something more fun, such as tetranglix from PoC||GTFO 3:8.

The `--cdrom CDROM` option in the PoC replaces any mapped CD-ROM with the provided image file.

The service provided by the application is possibly more interesting. A server administrator might connect a USB key to the system, perhaps containing a "kickstart" or "sysprep" file. The provided PoC will export the inserted Removable Media via NBD, which most Linux systems will happily mount as if it were a normal hard drive. This feature can be accessed with `--ndb` and `--ndblisten address:port`. Please be kind when testing, as this is exported read/write.

9.5 Have fun, stay safe

If you own a system that contains a BMC, please be careful what networks you connect it to, and which networks you access it through. A simple solution might be to connect a VPN device directly to it, and run a VPN client application on your desktop.

Remember that besides bolt-on security, such systems' management interfaces likely have plenty of other flaws. For example, see the SSH banner that the same BMC produces, or IPMI Cipher 0.