UNITED STATES DISTRICT COURT

CENTRAL DISTRICT OF CALIFORNIA

HONORABLE DAVID O. CARTER, JUDGE PRESIDING

\- - - - - - -

ECHOSTAR SATELLITE CORP., et al., Plaintiffs,	)
vs.	) No. SACV 03-950 DOC
	) Day 12, Volume II
NDS GROUP PLC, et al., Defendants.	)

REPORTER'S TRANSCRIPT OF PROCEEDINGS

Jury Trial

Santa Ana, California

Tuesday, April 29, 2008

Debbie Gale, CSR 9472, RPR
Federal Official Court Reporter
United States District Court
411 West 4th Street, Room 1-053
Santa Ana, California 92701
(714) 558-8141

EchoStar 2008-04-29 D12V2

APPEARANCES:

FOR PLAINTIFF ECHOSTAR SATELLITE CORPORATION, ET AL.:

T. WADE WELCH & ASSOCIATES
BY: CHAD M. HAGAN
CHRISTINE D. WILLETTS
DAVID NOLL
WADE WELCH
Attorneys at Law
2401 Fountainview
Suite 700
Houston, Texas 77057
(713) 952-4334

FOR DEFENDANT NDS GROUP PLC, ET AL.:

O'MELVENY & MYERS
BY: DARIN W. SNYDER
DAVID R. EBERHART
Attorneys at Law
275 Embarcadero Center West
Suite 2600
San Francisco, California 94111
(415) 984-8700
-and-
HOGAN & HARTSON
BY: RICHARD L. STONE
KENNETH D. KLEIN
Attorneys at Law
1999 Avenue of the Stars
Suite 1400
Los Angeles, California 90067
(310) 785-4600

ALSO PRESENT:
David Moskowitz
Dov Rubin

I N D E X

SANTA ANA, CALIFORNIA, TUESDAY, APRIL 29, 2008

Day 12, Volume II

(10:25 a.m.)

(In the presence of the jury.)

THE COURT: We're back in session. The jury's present. All counsel are present.

Counsel, thank you for your courtesy.

Mr. Stone, on behalf of NDS, your next witness, please.

MR. STONE: Thank you, Your Honor.

Defendants call Nigel Jones.

THE COURT: Thank you, sir.

Would you step between the double doors and raise your right hand.

NIGEL JONES, DEFENSE WITNESS, SWORN

THE WITNESS: I do.

THE COURT: Thank you, sir.

Would you be kind enough to be seated in the witness box to my left.

Sir, would you state your full name for the jury, please.

THE WITNESS: Yes, sir. Nigel Andrew Jones.

THE COURT: Would you spell your first name.

THE WITNESS: N-I-G-E-L.

THE COURT: And your last name, please.

THE WITNESS: J-O-N-E-S.

THE COURT: Thank you very much.

This is direct examination by Mr. Stone on behalf of NDS.

MR. STONE: Thank you, Your Honor.

DIRECT EXAMINATION

BY MR. STONE:

Q. Good morning, Mr. Jones.

Mr. Jones, how are you presently employed?

A. I'm president of R and B Consulting. It is a consulting firm I founded about 13 years ago.

Q. What line of work is R and B Consulting?

A. R and B Consulting provides design services in the field of electronics, software, embedded systems, and firmware.

Q. Were you retained as an expert by NDS in this case?

A. Yes, I was.

Q. What were you asked to do?

A. My primary role was to assess the large amount of technical information provided in this case and provide a technical forensic analysis of it.

Q. I'd like to talk a little bit about your qualifications and experience, sir. Can you tell us a little bit about your educational background, please.

A. I have a first-class honors degree in engineering from

Brunel University in London.

Q. What is a first-class honors degree?

A. Yes. In England they have a different degree classification system.

Honors is pretty much the same as in the United States, an honors degree. First-class honors is basically equivalent to a 4.0 GPA.

THE COURT: And would you state the university or college again?

THE WITNESS: Yes, Brunel, B-R-U-N-E-L.

THE COURT: Thank you very much.

BY MR. STONE:

Q. Do you currently live in England?

A. No, sir. I live in Maryland.

Q. And how have you been employed the last 25 years or so?

A. Basically my role in life is designing products. I design electronic circuits. I write the firmware that goes in the microprocessors that go in most of those circuits and provide those services to customers. So the chances are, several of you have probably at one time or another used something that I designed.

Q. Are you familiar with embedded systems?

A. Absolutely. My primary expertise is in the field of embedded systems.

To give you an explanation -- what are embedded systems? An embedded system is something that contains a computer but isn't a computer. So, for example, I have a little remote control here. Okay? It contains a microprocessor. This would be an embedded system.

The piece of equipment the court reporter is using also has a microprocessor. That would be classified as an embedded system.

Q. Have you written programs for embedded systems?

A. Hundreds. That's what I do for a living.

Q. Is a Smart Card also considered an embedded system?

A. Yes, it is. In fact, a Smart Card is just about the simplest form of embedded system you can have because it contains just one chip. Most embedded systems contain dozens or hundreds of chips. So a Smart Card is just a very simple embedded system.

Q. Can you give us an example of some of the commercial products that you've written programs for in the embedded systems area?

A. Yes, sir. Last year, one of my clients asked me to design a control system for a diesel burner that is used by the United States Marine Corps on their mobile kitchens. The problem that the Marines were having is that the controller on the existing one would get corroded in adverse environments and fail, and then they couldn't cook. And a hungry Marine is an angry Marine.

And they came to me and said, "Can you design us a better mousetrap, a better control?" And going off, I designed all the electronics for it, wrote all the firmware. We did the first production run right at the beginning of this year. The Marines have seen that product, and they can't wait to get it in the field.

Q. Have you written any software for scuba diving equipment?

A. Yes, I have. One of the more interesting areas I work in is, in fact, scuba, particularly highly advanced diving systems. If you've ever seen anything on the Discovery Channel where you've got divers doing really neat stuff down deep, there's a really good chance they're wearing something that I've designed.

My latest product that I'm working on for a company in Sweden was just featured in Popular Mechanics last month.

Q. Do you hold any patents?

A. Yes. I have one patent issued and quite a few pending.

Q. And what are those fields in?

A. Yes. The patent that's being issued relates to a smart battery that the U.S. military uses in all their equipment. So I came up with a rather neat way to help the U.S. military extend the use of those batteries. So that's the patent that's issued.

Patents that are pending relate to this control I just mentioned to you that I designed for the U.S. Marine Corps. I also have some other patents pending on the diving stuff.

Q. Have you written any articles in the embedded systems area?

A. Yes. In the embedded systems area there is the premiere magazine called Embedded Systems Design. I've written about a dozen articles for that magazine. I'm also on the editorial design review board. What that means is, when an article is submitted for publication, if the editor thinks it's in a field that I have particular knowledge of, that paper will be submitted to me for vetting or approval.

Q. And have you had any experience in assisting any companies as an expert in satellite piracy?

A. Yes, I have. Four or five years ago, I was retained by DirecTV, along with my colleague, Mr. Barr, who's in the back here, who you'll be hearing from in a few days. The two of us, plus a couple other gentlemen, spent the best part of actually more than a year examining the hundreds of devices used for DirecTV piracy. And so our job was to take these devices, reverse-engineer them, work out what they did, how they did it, and come to a conclusion whether those devices were compatible with, designed for, suitable for DirecTV piracy.

So having spent -- I think it was about a 15-month period for me looking at all these devices, I learned a tremendous amount about satellite piracy, how it's done, the different devices that are used, and so on.

Q. Have you ever been retained by Bell ExpressVu or EchoStar in connection with a satellite piracy case?

A. Yes, I have. About two or three years ago, there was a joint raid in Canada between DirecTV, EchoStar, and Bell ExpressVu. And what they were doing, they were going to raid a printed circuit board manufacturing plant. And this was a place that was suspected of making printed circuit boards used in all these DirecTV and Bell ExpressVu and EchoStar piracy devices, so they needed someone who, (a), knew what the devices were, what they looked like; and they also needed someone who designed printed circuit boards and knew their way around a circuit board plant. So I went off on this raid. Very dramatic -- police, lawyers turn up at the door, stand back from the desks, and then they bring the engineer in. A fun experience, actually.

Q. Do you have any experience with the microprocessors used in the EchoStar access cards?

A. Yes, I do. Microprocessors come in families. Okay. And the family of processors used in the Smart Card issue in this case generically is called a 6805. The 6805 was almost the first microprocessor I ever programmed, and I've written hundreds of programs for the 6805 and other members of its family.

Q. Have you also worked with encryption in your design work?
A. Yes, I have. I use encryption in two ways. A lot of the products I design include what are call encrypted bootstrap loaders.
I also have a client that is in the car wash industry. And the car wash industry, as you know, when you go up to the car wash, you have a machine there where you can pay, and they'll take cash, credit, or debit. Well, with debit cards, Visa or MasterCard have a very stringent set of encryption stuff that you have to go through in order to have a debit keypad on a car wash system. If you go to Wal-Mart and go through their car wash, and you pay with a debit card, you'll be using software that I wrote. So if you've done it, I hope it worked.
Q. How many programs have you written total for various microprocessor families?
A. Oh, hundreds and hundreds. I've been doing it for 25 years. It's what I do every day.
Q. Have you ever done any reverse engineering?
A. Oh, yes.
Q. Is reverse engineering a common practice?
A. Oh, yes.
Q. Is it a secretive practice?
A. No, not at all.

Q. Can you give us any examples of that?
A. Oh, yes, absolutely. I'll give you two examples.
Whenever, say, Toyota brings out a new car, the first person to buy that car is General Motors. What does General Motors do? They reverse-engineer it. They rip it apart, they look inside, they see what it cost, they look for new technologies, they look at things they think Toyota was doing badly.
A second example, which is much closer to home, last week -- I mentioned Embedded Systems Design magazine that I was involved with -- twice a year that magazine holds a conference. The main conference is in Silicon Valley. Unfortunately, this year it coincided with this trial, so instead of being at the conference, I was here.
But at that trial -- excuse me -- conference, the advertising literature for the conferences -- one of the highlights of the conference was going to be they were going to tear apart the latest Sony OLEV -- stands for Organic LED TV -- this is the next big thing in TV.
So at the conference as a draw to bring people in, "Come on in. We're going to take this thing apart." I don't think that's very secretive.
Q. Is reverse engineering a widespread practice in your industry?
A. Yes, it is.

Q. And how much time have you spent on this case?
A. Hundreds of hours. I think I'm up to about 800 hours.
Q. And what have you done with that time?
A. Quite frankly, I've almost gone bug-eyed. I have looked at thousands of files, many of which are what are called binary files. When you think of a file, most of the time you think about text, okay? That's what you read. Well, computers also use binary files. So I've had to look at, at least, a hundred, probably more, binary files, which means I have to take those binary files, put them into special programs to allow me to examine them. I've gone through all these different files. I've gone through what I refer to as the Conus e-mails: Six-and-a-half ring binders, each one that thick.
THE COURT: Six different --
THE WITNESS: Six different.
THE COURT: Conus?
THE WITNESS: C-O-N-U-S.
THE COURT: Conus e-mails.
BY MR. STONE:
Q. And those would be e-mails reporting on the status of the DNASP-II system?
A. Correct.
Q. Have you reviewed other documents in this case as well?
A. A tremendous number of other documents, yes.

Q. And when did you prepare your original report in this case?
A. I started work on it in March of 2007. The report was submitted May 10, 2007.
Q. And were there any significant documents that came to light after you did your original report?
A. Yes. Two major sets of documents come to mind. The first one is what I refer to as the "black box files." And the second set was the source code to the DNASP system.
Q. And when did you have an opportunity to review the source code?
A. About two weeks before the trial started.
Q. And did any of those new documents change any of your opinions?
A. No. Actually, they did quite the opposite. The documents confirmed a lot of things that I suspected. If anything, they helped to confirm my opinions.
Q. And you've also reviewed the deposition testimony in this case?
A. Yes, I have. I've probably read at least eight deposition transcripts.
Q. Now, based on your review of the evidence and the documents and the files that you've testified to, were you able to reach opinions in this case?
A. Yes, I have.

Q. And have you prepared a demonstrative that summarizes your key opinions?
A. Yes, I have.
MR. STONE: Can we show that to him.
(Document displayed.)
THE WITNESS: This is my first opinion. Haifa was not the source of the Nipper postings nor any of the other information on the Internet.
BY MR. STONE:
Q. When you say "nor the -- any of the other information on the Internet," what are you referring to?
A. Well, other than the Nipper postings that are issued in this case, we also have a lot of other technical information: the StuntGuy FAQ, Dover FAQ, and hundreds of other bits and pieces of information that are cropping up. Perhaps what's most germane are excerpts of ROM from the DNASP system.
Q. And we'll be going through that information, and you'll have an opportunity to explain why you concluded it is not connected to the Headend Report, correct?
A. Correct.
Q. And have you formed any other opinions?
A. Yes. This is my second opinion: My second opinion is that it was inevitable that the NagraVision system would be hacked.

Q. You formed any other opinions?
A. Yes. My third one: NagraVision knew about the problems in their system before the Nipper postings and chose to do nothing about them.
Q. And do you have a final major opinion?
A. Yes. The patch that NagraVision applied to the card within months after the Nipper postings was completely effective in closing the buffer overflow vulnerability that you've heard so much about.
Q. Was there also an electronic countermeasure that accompanied the patch?
A. Yes, There was.
Q. Have you studied that as well?
A. Yes, I have.
Q. Have you studied both the patch code as well as the electronic countermeasure information?
A. Yes, I have.
Q. Okay. Now, let's talk a little bit about the EchoStar satellite system. And the folks on the jury have heard some, so I'd ask that you give a very brief review.
A. Brief?
Q. But slow.
A. I'll try. Okay. What we have here is a basic picture of how this all works. I think you've got the basic gist.
We have a big satellite uplink dish. It sends an

encrypted signal to the satellite, gets bounced off the satellite down to your satellite receiver dish. That signal goes into the receiver. The receiver basically says to the Smart Card or access card, "Does this person have permission to see the particular channel?" And if they do, you get to see Shrek on your TV. Okay? So that's the basic way this thing works.

What I'd like to do now is show you the thing that's really at issue in this case. This is the receiver, so there's a little bit more detail here.

What we have here is messages in encrypted video coming in, and the messages get routed to the access card. The access card has got a couple of components that I'll talk about in more detail later. The access card, if the person is authorized, provides the encryption key, and that encryption key allows video.

So to show that happening here, we've got messages in encrypted video coming in, keys being provided by the access card, and bingo! Decrypted video.

Q. Okay. And do you have an animation of the normal operation?

A. Yes, I do.

Q. Okay.

A. So this is an animation that I'd like you to look at. And before we play it, I'm gonna explain a little bit about what you're going to see, and hopefully this will make a little clearer to you a lot of these buzzwords you've been hearing thrown around for the last few weeks.

You have a remote control, and the remote control is gonna ask, "Please, can I watch HBO?" Okay. The message is gonna go to the receiver, receiver is gonna put that message into the I/O buffer. This is the buffer you've heard about that's being overflown all the time. Okay?

This little thing here represents the buffer filling up, and you're going to see this a few times today. Okay? The man in the middle is the CPU. That's the actual microprocessor that's doing the work.

There are three what I call "sets of books" in this Smart Card. Over here we have the EEPROM. Now, the EEPROM contains things like decryption keys, passwords, pay-per-view authorizations, and so on.

In the middle, we have user ROM. These are the general instructions for the CPU in terms of what it must do.

And then we have the system ROM, which is really responsible for encryption-related functions.

Lastly, we've got this funny little guy here called a "guard." This is representing what is called the "memory access control matrix." The memory access control matrix is going to feature quite heavily in my testimony. The way to think of it is, it's a security guard.

This is the guy with his hands like this. Okay?

THE COURT: Okay.

BY MR. STONE:

Q. And can we roll it?

A. Yes, sir. Can we run the animation, please?

Here we have the HBO request coming in, goes to the receiver, into the I/O buffer. You see the I/O buffer filling up. CPU goes along, picks up the message, says, "Okay, what do I do with it?"

It goes to the user ROM to get the general instructions and goes to the EEPROM and says, "Has this dude paid for HBO? If he has, give me the decryption key."

He then takes that decryption key over to here. But first off, the security guard says, "Hang on, pal. Are you authorized to do this?" So he checks the credentials, allows the guy in.

What this guy does now is, he takes the key and puts it in a lock box. And it's going into a lock box because that key's secret, and you've got to transmit it back to the receiver in a secret way. So he's put it in a lock box. It goes in over to the receiver, receiver extracts the key, and guess what? We've got Shrek on TV again.

Okay?

Q. Can you give a sense to the jury how long that whole process takes in real time?

A. Yes. In real time, we're talking about maybe half a second. Okay? And I might add as well the most time-consuming portion is this bit around here. Okay?
Q. Okay. Now, do you have an animation that illustrates what we've heard a lot about, called the buffer overflow attack?
A. Yes, I do.
Q. Can we run that?
A. Right. So in this case, things have changed a little bit. We no longer have our receiver. We've got what is called a Smart Card reader/writer. And attached to that is the hacker's computer.
What I'm going to show now is what happens when a hacker sends a message that is bigger than the buffer, and that buffer overflows.
So if you could run the animation, please.
Here's our big red message. Okay? Comes into the I/O buffer. The buffer starts filling, and then it overflows.
Now, the way to think of this message is like a computer virus. Okay? So the CPU goes and picks it up, and he is somewhat confused. I mean, this is something he doesn't quite know what to do with. So the virus takes over and starts commanding the CPU what to do.
In this case, what's happening is, he's modifying the EEPROM and also getting the EEPROM contents going over to

the I/O buffer, sticks it in the buffer, which then goes out to the hacker's computer.

So at this stage, okay, that is a buffer overflow in practice in this card. Okay? And by doing this, the EEPROM contents are now available on the hacker's computer.

Q. Now, we've talked about two Internet postings in this case. And the first one I'd like to focus on is the December 23rd posting by xbr21 of something we've been calling the Nipper code.

Did you do anything to determine if that Nipper code came from the information in the Headend Report?

A. Yes, I did. I ran an extensive amount of analysis on it.

Q. Can you tell the jury in broad terms what you found when you compared the Nipper code to the Headend, or Haifa, Report?

A. Yes. So I looked at this program in many ways, and what I found in broad terms was that wherever Haifa and Nipper had a choice to do something, they chose differently.

Q. Surely they must have made some of the same choices?

A. Actually, no, nothing substantial.

Q. Now, plaintiffs have identified four things that they claim prove that the Headend Report shares the same DNA, I think was the reference, as the Nipper code. What is your opinion on that?

A. Yes, this is very interesting. You heard Dr. Rubin talk about what I consider the four pillars. He identified four things.

So Dr. Rubin identified four pillars that he said were characteristic of an attack that must have originated from Haifa. What I will be showing you is that any buffer overflow attack on this card must use those four things that Dr. Rubin identified.

Q. Now, there is also testimony that any differences between the Headend Report information and the Nipper code is a result of two years' time to improve the Headend information.

Do you have an opinion on that?

A. Yes, I do.

So what Mr. Stone is referring to here is that the suggestion that any differences between Nipper and Haifa can be attributed to the two-year difference in time between when Mordinson wrote his code and when Nipper wrote his code. Well, what I'm going to show you is that David Mordinson's architecture, the way he put his program together, is considerably better, superior to what the Nipper architecture is.

Q. So there was no improvement in that intervening time period?

A. No, the exact opposite.

Q. Now, can you explain to the jury the steps you went through to compare the Nipper code to the Headend Report information?
A. Yes, I think so. I'll start with the next slide.
What this slide is, this is the Nipper posting, okay? And over here on the left are his instructions, and this was literally what was published on the Internet. And even if you could read it in detail, you'll see that it is just a bunch of hexadecimal numbers. Okay?
So, because this was supposedly derived from Haifa, the first thing I did was say, "Okay, let's look at David Mordinson's equivalent." So on the left is Nipper code. On the right, this is taken from Appendix "F" of the Headend Report.
I believe you have this in evidence. You can go and look at this. Okay? On the left, Nipper; on the right, this is David Mordinson's code.
Q. Did you do anything to make it easier to compare the two?
A. Right. Well, for those of you that know anything about computer programming, you will recognize what's on the left here is what's called a binary representation, and this is what's called source code. So to do a comparison, obviously I have to convert the two into the same format. I started off by converting them both to binary.

Q. What does this show here?
A. The Nipper code and the Mordinson code, side by side, now in the binary format.

Okay. Now, you don't have to know anything about computer programming or chips or whatever to see immediately that the Mordinson code is a different size. Okay? Well, so what about the actual values that are in it? What I did is, I said, "Well, I put the two programs on top of each other, and the red is where codes don't match, and the gray is where the codes do.

And so to further illustrate the point, I've now removed all the places where they don't match.

So it -- what this illustrates here is evidently there wasn't much of a match between the two at a binary level. Okay?

Now, if you've done any computer programming, you will know that you need to make a very slight change to what's called the source code to make the binary image completely different. So this is perhaps not a particularly fair comparison, but it is an interesting one nevertheless.

What I did now was, I went the other way. I converted Nipper's code into what is called source code. I did what is called disassembly. So now Nipper and Mordinson are in exactly the same format, but this time in a source code format.

Q. And what did that show?
A. Well, I'm sure you can't see enough detail on the screen there, ladies and gentlemen, to really be able to tell, but if you went up and looked at what are called the actual operation codes up there, you would see that these two programs are completely different. They differ in many, many ways.
Q. Can you tell the jury some of the other ways in which the programs differ?
A. Yes. So the first thing I'd like to show you is this line that I have highlighted here. Now --
Q. And what is that?
A. The line I have highlighted here is the call to transmit a byte out of the card. So if you remember, the purpose of this program is to read the contents of the EEPROM book and transmit it out. And it transmitted out a byte at a time. Okay? Now, if you look carefully, you will see that the subroutines that are being called by Mordinson and Nipper are very different. Let me show you how different.

Nipper chose to use a routine that was built into the card. Okay? He took one line of code to do it. David Mordinson, by comparison, decided to write his own routine. He devoted 36 lines of code to do what Nipper did in one line. To me, that's a very fundamental difference in the way the two people were thinking.

Q. Okay. Was there a difference in the way they terminated the programs?

A. Yes. We've gone back to the slide here, and what I have highlighted here is how the program ends. If you look on the bottom, that is David Mordinson's code. And you see this very strange thing that says "B-R-A-$." That means in assembly language branch always to yourself. In other words, loop on yourself. Okay? Go into an infinite loop.

Now, there's only one way out of an infinite loop, and that is to reset the card. Pull it out, plug it back in. Not a very elegant way of finishing a program.

Q. Is that consistent with something called "proof of concept"?

A. Absolutely.

Q. What is proof of concept?

A. Proof of concept is something I get to do all the time. It is -- customers come to me, and they say, "We've got this great idea that we think we can turn into a product. We're not sure it can work. What we want you to do is just enough work to show that the concept is good. Prove out the basic ideas. We don't want fancy code. We don't want it well documented. Just do the smallest possible amount of work to prove it out." That is called proof of concept.

Q. How did the Nipper code terminate?

A. I think this was very interesting. The Nipper code jumped to location 7381.
Q. What does that mean?
A. What that means is, it is jumping into part of what is called the user ROM. And furthermore, this jump requires you to pass what is called a parameter. Okay?
You'll see that strange notation, ".DBE8." That is a parameter being passed to that subroutine. Now, here's the rub. That subroutine 7381 does not appear in the Headend Report. So the person that wrote this code must have had something else other than the Headend Report. What they must have had is the ROM contents. Okay? If they had the ROM contents, they could do exactly what David Mordinson had done.
Q. Are there any other differences that you saw between the two programs?
A. Yes, many. This slide here that I have, the first four we have already discussed. So program size, the actual detail of the coding sequences, the write routine used, how they terminate the program.
The fifth one is kind of easy to explain. You've heard some testimony about invalid checksums before. Well, the interesting thing is that Nipper and Haifa chose to use a different value for the invalid checksum.
Q. What is the next point?

A. The next three points -- stack pointer, addressing use, how it handles interrupts -- quite frankly, ladies and gentlemen, you need a degree in electrical engineering with computer science to understand those. I'll just ask that you accept -- when I tell you they are significantly different, that you accept that.

THE COURT: The jury will understand everything, both you and the other expert.

BY MR. STONE:

Q. That means manipulate and interrupt, just briefly, versus not using an interrupt?

A. Yes. So what an interrupt is, in an embedded system is, when the program is running normally, and then something happens that causes the program to stop doing what it's doing and run off. It's a bit like when you're working at your desk, or whatever, and the phone rings. The phone is an interrupt. Okay? You handle the phone call, you put the phone back down, and then hopefully you carry on the work you were doing.

Okay. So interrupts feature very heavily in embedded systems. They're one of the most difficult things to grasp and do correctly. So the fact that the authors of these two codes took different approaches to the use of interrupt handling is highly significant.

Q. And in broad terms, what is the difference between

direct addressing and indexed addressing?
A. With direct addressing, you say, "Give me the value from this specific location." With indexed addressing, you say, "Give me the value at a location that is offset from a base address."
Q. If you could move the pointer -- there you go -- there's a reference to "shell code" in the second column from the bottom.
What is shell code?
A. You've heard the term "shell code" before. I think a better term for it that's easy to understand is "virus." This is the virus that we're putting into the card to take it over and do its thing.
Now, what's important here is where Mordinson located that virus and where Nipper did. Mordinson located it in the communications buffer. Nipper located it in a region called the stack. And I will be showing you later why that is incredibly significant.
Q. Do we have a slide for that?
A. So what this shows here is, we have the Nipper code and the Mordinson code side by side again. And the light blue that you're looking at, that is David Mordinson's shell code. And you can see it's at the top in the communications buffer, whereas the Nipper code is at the bottom in the stack region.

And I've just gone ahead and highlighted those areas.
Q. And just briefly, what is the significance that you'll be talking about for that difference?
A. Basically, by David Mordinson putting the program, the virus, into the communications buffer, it allowed him to deliver a program faster that was bigger and was easier to use. And that will be the basis of my opinion that the Mordinson method is superior to the Nipper method.
Q. Okay. Now, if we could go back to the slide or go forward to the slide with the summary. Okay, if I understand it correctly, there were at least those 10 differences between the two programs?
A. That's correct. There were many more, but I felt that 10 was more than enough to make my point.
Q. And how would you summarize these differences?
A. To me, when you look at all these differences, it is clear that these two programs were written by different people independently. I see this as independent development of these two codes.
Q. So do you think the Headend Report was the source of the Nipper code?
A. No, I do not.
Q. Now, did Dr. Rubin disagree with you on the point that these were different programs?
A. No, no, he didn't, actually. I had this excerpt from

his expert report, and this is what he has to say: "The point of contention here is not whether or not the two programs are the same, because clearly they are not."

Q. Now, did you find any significant error in Dr. Rubin's report that might influence the assessment of his method of charting the structure of the two programs?

A. Yes, I did. In Dr. Rubin's report, he put together some graphs which showed the two programs, and he used those graphs as a basis or an aid to reaching his opinions.

Unfortunately, there were some errors in those graphs which I think are quite significant.

MR. STONE: Okay. And, Your Honor, may I approach the easel?

THE COURT: You may.

BY MR. STONE:

Q. Mr. Jones, what I'd ask that you do is step down and demonstrate how you found the error and what the consequence of that error is in the analysis.

For the record, we have two blowups of Page 35 and 36 from the Appendix "F" of the Headend Report.

THE COURT: Just a moment. I want to see if Dr. Rubin can see also.

DR. RUBIN: Yes, I can.

THE COURT: If you need to get closer, either expert, that's fine.

THE WITNESS: Okay. Ladies and gentlemen, what I'd like to show you is this excerpt from the Headend Report, which is David Mordinson's code. You can go into the jury room and look at this and do what I'm about to do just for yourself. Okay?

The way to look at this is, in this column here, these numbers here are the numbers that appear over here. Okay? These are what we call pneumonics or the actual op codes that the computer executes. These are variables associated with those op codes.

And over here we have comments. A typical comment, load to high byte, check the EEPROM boundary, and so on.

What was done with Dr. Rubin's report is, he looked at this and started at the top, and he saw "2100A8." And that's what you see here.

BY MR. STONE:

Q. So these three bytes match. And then we come down to this byte, and you see a whole series of 9Ds. And here you have a whole series of 9Ds. And we come all the way down to here, CC01A0. And that's these three bytes here.

And you notice that Dr. Rubin then says that is the end of what he calls the shell code, or the virus.

Well, he didn't look below the line. This is a subroutine here that's very important, and you can see the subroutine now. We get all these 9Ds, and then we come to 1100, and, in fact, we go all the way down here, all the way up here, and all the way down to the last 81, which is here.

So in reality, the shell code isn't here; the shell code includes all of this.

Now, I thought that was quite significant. We're not talking about a few bytes here. We've missed well over half the program. And the importance -- that I can now show you on an animation.

Could we step to the first part of the animation, please?

So what you see here, this is exactly the same thing we just had on the board. Okay? This is an excerpt from Dr. Rubin's report. All the labels and things are his. Okay?

We'll now step the animation, please.

All I've done now is add color. Okay? And what I want you to understand is that the different colored regions do different things. Okay? I haven't changed anything, just colorized it. So as this shows right now, the light blue is where Dr. Rubin says the shell code is.

We step the animation, please.

So what I've done now is go ahead and correct the shell code representation. As you can see, it's quite dramatic.

Can you step the animation, please.

As you see here, Dr. Rubin has identified this yellow area as what he calls overflow. It isn't overflow. There's some padding in there, but there's setup of some very important variables in low memory which will become important later.

Step the animation, please.

So what we're now showing, the stuff in red is what David Mordinson considered important setup of that memory location. The stuff in yellow are padding bytes. Okay? So this represents a much more accurate and detailed representation of David Mordinson's code.

When I realized there were mistakes in the Mordinson representation, I asked myself, well, is there a similar problem with the Nipper representation?

And I have an animation that shows that. So this one's a bit shorter. So same thing, this is from Dr. Rubin's report, and the first thing I do is colorize it.

Now, the thing I want you to know is the colors I've added are consistent. So the shell code is still light blue, overflow is in the same color, and so on. Okay.

So can we step the animation, please.

Again, Dr. Rubin had a bit of an error in his description of these overflow bytes. And what you see here is the yellow is indeed padding, but the red is what Nipper considers to be important memory setup.

Okay. So what?

So can we go to the next animation, please. Oh, I'm sorry. I hadn't quite finished that. We had a terminology problem as well that got corrected.

So what I've done now is I've put these two side by side for you to see the bigger picture. So I've dispensed with the monochrome version, and I've gone straight to the colorized version. So what I'm going to do is correct the errors one by one.

Can we step it, please.

So there's the shell code.

Next, please.

That's the stack setup.

Next, please.

That's the correct representation of the Nipper code. Now, to show the significance of that, could we go back to the first step of that, please.

That's where we started.

Now -- I'm sorry. Can we stay on the first one, please.

I don't know about you, but if you don't know much about computers, you just look at those two pictures and you say, well, yeah, they're basically the same. You just move the light blue up to the top and you've got the same thing, right?

Now, one thing I must stress here, ladies and gentlemen, is that the data within each of the colored regions are different. I'm not saying that these two blue regions are identical. They're not. It's simply their basic function we're talking about. So that's where it started.

Can we go to the end now, please.

I think that's a considerably different representation of the two programs.

BY MR. STONE:

Q. Mr. Jones, there are some colors that are the same in the same place. Like up at the top there's a white box.

A. Yes, sir.

Q. What does that represent?

A. Right. So in some cases -- that white box, for instance, is the ISO7816 mandated header. In other words, the international standards say you've got to have that there. You have no choice.

Q. Are there any other no-choice areas between those?

A. Yes. You can surely have noticed that towards the bottom third there's this dark blue region. You've all heard about the buffer overflow and memory aliasing. This is that buffer overflow region where you have no choice in the matter. So the hardware in the card is dictating what you see there. You can do nothing about it.

Q. So once again, where the authors had a choice, did they make completely different choices?
A. Correct, they did.
Q. What's the significance of that?
A. Well, to me, if you have got two people making completely different choices wherever they have a choice, the logical conclusion is they developed these things differently, independently. There was no cross-coupling between them.
Q. Now, the next area I'd like to shift to is the plaintiff's claim that were four characteristics between Headend Report and Nipper that show that they share the same DNA, the four pillars, as you've described them.
A. Yes. I think we have a slide here that shows my understanding of what Dr. Rubin said.
Q. And those four things are the use of a buffer overflow attack, the use of memory aliasing, knowledge and use of the index variable, and knowledge and use of the exception handler, correct?
A. Correct.
Q. And if you analyzed each of those four pillars, as it were --
A. Yes, I have.
So let's take the first pillar. The claim is, because the buffer overflow attack was used, this is indicative it

came from Haifa. Well, you've heard testimony from Mr. Nicolas and Dr. Rubin that a buffer overflow attack is the most common form of attack on any computer system. That was true back in 1980; it was true in 1990; it's true in 2000; it's still true today. Okay?

If you get this little update from Microsoft that says "Windows has been updated," there's a good chance that they've just patched a buffer overflow vulnerability in their code.

Q. The second item is memory aliasing. Can you explain a little bit about memory aliasing?

A. Memory aliasing is a strange topic. And so I have some slides here which I hope will help you better understand what memory aliasing is.

So consider this: We've got ourselves a street, Memory Lane, and on there we've got four houses. And our person, our mailman here, is going to deliver a letter addressed to 120 Memory Lane. And you can see the mailman has absolutely no difficulty in doing it because 120 Memory Lane is there, and the letter will be delivered.

Well, what happens if you send a letter to 180 and relay? The mailman doesn't know what to do with it, so he will mark it as "Return to Sender, Not Deliverable."

Okay. But what happens if you get a letter addressed to 220 Memory Lane? The mailman could do one of two things here. He could say, "You know what? 220 doesn't exist. I'll mark it 'return to Sender.'" Or he could say, "You know what? I bet they meant 120 Memory Lane, so I'll deliver it to 120 Memory Lane."

That, ladies and gentlemen, is memory aliasing, where something designed for one address gets sent to another address.

Q. And why do chip manufacturers allow memory aliasing to occur?

A. Fundamentally, it's a cost-savings measure.

I'll have to explain a little bit about how chips are designed. When you design a chip, in there you build in something called a memory management unit. And the memory management unit, as its name suggests, is a piece of a chip whose job it is to manage memory.

Now, when you design a family of microprocessors, you typically design the memory management unit such that it can address, say, this much memory.

Now, if you don't need that much memory in your chip, say this much or this much, they don't actually change the memory management unit. They just say just don't install so much memory. And that is exactly what happened on this chip. Okay?

Q. Is there an easy way to tell if a particular chip is memory aliasing?

A. Potentially. Sometimes you can just read it in the data sheet. But regardless of that, the easiest way is to run a test, do the equivalent of send a letter to 220 and then go to 120's mailbox and see if it got it. It takes you two, three hours tops to run that test.

Q. Is memory aliasing used a lot in the industry?

A. Certainly when I first graduated, first 10, 15 years, yes, it was used a lot. It was almost standard. Today, with the way they design chips differently and so on, it's becoming less and less common.

Q. Now, is memory aliasing something that would be a unique characteristic in the buffer overflow attack on this particular chip?

A. Ah. Well, here's where it gets interesting. Okay? As soon as you write beyond the end of the communications buffer on this chip, aliasing occurs. Okay? You have no choice in the matter. You can't say to the chip, "This byte I'm writing I don't want you to alias. This byte I would like you to alias."

It happens regardless. You have no control. So what's the implication of this? The implication of this is, if you perform a buffer overflow attack on this card, you have to exploit aliasing because you have no choice.

Q. Well, just to be clear, is there any way to perform a buffer overflow on this card without having the data memory

alias beyond the buffer?
A. No.
Q. There was a third point, the use of the index variable.
A. Yes. We've heard a lot about the index variable.
THE WITNESS: What I'd like to do now, Your Honor, with your permission, is to go in front of the jury and do some drawings.
THE COURT: Certainly.
THE WITNESS: Okay. What we have here is what engineers like to call a memory map. So I'm going to give you an analogy first so you can better understand it. Imagine you've got a big apartment building with lots of residents, and all their mail is delivered in mailboxes at the bottom of the building. Okay? Each resident has one mailbox.

Think of this as all the mailboxes for that building. Okay? And just as you can say -- refer to, say, the tenant who's in apartment 3C, or you can say Mrs. Smith, so you can with memory.

So our famous index variable -- we can either refer to the index variable or we can refer to its address or, if you like, the apartment that it's in. Okay. In this case, apartment 3F.

Now, Dr. Rubin was kind enough to explain hexadecimal numbers to you. So even though these numbers look a little strange for what we are used to counting in, they are real numbers.

And so what I want to show you here on this picture -- this is the memory of the microprocessor in this card. Now, at the bottom we have what are called the registers. We then have some memory. We get our index variable, a location called top of stack. And then up at location 19C it's the start of the I/O buffer. This is the famous buffer that is being overflown. Okay?

What I'm going to do now is show you how the index variable fits into all this.

So when a message is received, okay, start of a message is received, the code in the user ROM -- remember the instruction book on the animation -- is going to store the first bytes it receives in the I/O buffer at the offset given by the index available.

Wow! That was a bit of a mouthful.

So let's put it in practice. When the message first comes in, the index variable has the value zero. And so what will happen is the first byte will come in. It will be stored in the I/O buffer at offset zero, here, the first location.

Okay. We then increment the index variable to 1. The next byte that comes in will be stored in the I/O buffer at the offset given by the index variable, offset 1.

So far so good.

Now, the I/O buffer is a hundred bytes long, and so I think you can see that by the time you get to the end of the I/O buffer, the index variable is going to have a value of 99. Okay?

But what happens now? Well, the next byte we send gets aliased; that is, it comes off the end here, all the way down into here.

Well, it so happens this region called registers is special. Okay? It can't be touched by the aliasing.

And so you keep on sending bytes, and nothing much happens until the index variable has got a value of 132. At that point you are now into memory. So what that means is when your index variable is 132, the next byte that is received gets stored here. Okay.

Well, I think you can see what's coming here, folks. As we overflow the buffer more and more, when the index value -- variable has a value of 162, we're here, just below the index variable.

Now, Dr. Rubin says if we use the index variable, the attack must come from Haifa. So to not use it, at this point I stop.

Anyone have any idea what will happen at this point? Well, I understand you're not allowed to answer my questions, and I don't want to upset the judge here, so I'll answer the question. Okay? The answer is nothing happens, or nothing substantial.

Okay. We've gone to all this trouble of overflowing the buffer. We've got all the way to here, and nothing happens. Well, it seems to me, then, we don't have any choice. We need to overwrite the index variable. Okay.

So the next byte we send overwrites the index variable. Well, what happens if we overwrote the index variable with zero? Where would the next byte go? Well, the next byte is stored in the communications buffer at the offset given by the index variable. It goes right back there.

Well, that's not very useful. In fact, if you chose any value of the index variable between zero and 162, you just end up somewhere around here, and you just go in an infinite loop all day long. Okay? Patently not very useful.

So you have to do something with this index variable. Well, so what did David Mordinson do? What David Mordinson did is, he said, you know, "I'm gonna modify the index variable such that the next byte gets written to the top of stack." Okay? That's what David Mordinson did.

Well, what did Nipper do? What Nipper did is he modified the index variable to point to here. Well, why did he point it there? Because that's where his program wanted to go. And his program was just big enough such that the last byte of the program overwrote the top of the stack. Okay.

Well, you've also heard about black box. What did black box do? Well, black box modified the index variable such that the next byte got written immediately thereafter it. Okay.

So all three of them modified the index variable, but they modified it in different ways.

Now, the question is: How did David Mordinson know how to modify the index variable? I presume he didn't call anybody up. Okay? So the only way he worked out how to use the index variable was to take the ROM contents, study them, work out how the program works, and go for it.

I might add, this whole use of index variables and things is a very standard procedure. Okay? There's nothing complicated or difficult about it. If I was implementing code like this, this is exactly how I would do it.

So David Mordinson worked it out by reading the ROM.

Well, does that mean that Nipper could have worked it out by reading the ROM? Of course. And also the same for black box.

So the point I want you to take away from this, ladies and gentlemen, is I see no way of constructing a buffer overflow attack against this card which doesn't use the index variable. It is impossible.

Q. Now, the fourth point was the exception handling. Can you explain that?

A. Yes. So far the three steps we've talked about, all they do is get the virus into the card. Okay? But it's dormant at this stage. It can do nothing.

And so we have to somehow persuade the CPU, the microprocessor, to activate that virus. Well, it turns out that there's a fairly standard way of doing this, and that is what's called forcing an exception.

As its name suggests, what an exception is, is when you cause something to happen to a computer system that is unusual. Okay? Now, when you write these programs, you like to take care of all contingencies, and you build in what is called an exception handler whose job is to handle exceptions.

And so the interesting thing about this card is, is the exception handler was designed in such a way that if you were to put a pointer to your shell code at the top of the stack -- remember my picture we had of the top of stack -- if you put the pointer to your shell code there and then force an exception, then the exception handler will end up running your code or, to put it in the vernacular, activating the virus.

So the question is: Is this unique?

Well, I've studied the code. I've gone through all the documentation and so on. And as far as I can tell, there are only two ways of generating an exception that would cause that behavior to happen.

The first is to send an invalid checksum. Now, I think you've had this explained to you before. Checksum is nothing more than where you sort of add up all the bytes that have gone beforehand, and if they equal what you've previously got, there is no error. So you deliberately send an invalid checksum and you say there is an error in the message. So that's one way to do it.

There is another way. You don't send a checksum. And what happens then is the card says, "Hang on a sec. Where's my checksum?" And it will wait for a large fraction of a second, eventually decide it's not getting the checksum, and run the same exception handler.

So two ways of doing it. Sending the invalid checksum is obviously superior because you don't waste the time waiting for the timeout. And so what we have here is, in my opinion, the only way of activating that virus in this card is to exploit the exception handler by sending an invalid checksum. There is no other way.

So if there is no other way, it is not surprising that a buffer overflow attack that was independently developed

exploited that characteristic.
Q. So taking these four pillars, the buffer overflow attack itself, that's the most common form of attack on computers?
A. Correct.
Q. Would anyone engaging in such an attack have any choice as to memory aliasing for this chip?
A. No, they would have no choice at all.
Q. Would there be any choice about using the index variable to engage in such an attack?
A. Not that I can see.
Q. And for the exception handling, would there be any choice as well?
A. None.
Q. So memory aliasing, modifying the index variable, and exception handling are all necessary structural features for a buffer overflow attack?
A. On this particular chip, yes.
Q. Now, when you discussed the index variable and the exception handling, you mentioned that you could deduce those from having the ROM contents, correct?
A. Correct.
Q. Did you see any evidence that the ROM contents from satellite cards were obtained by pirates?
A. Yes. There's a tremendous amount of information

supporting that position.

Q. Do we have a demonstrative to start off with?

A. I think so.

What we have here, ladies and gentlemen, is a historical perspective. What this is, is a list of all the Smart Cards for satellite TV that I know about that had their ROMs extracted by pirates.

As you can see, it's an extensive list. It starts 1993 and goes all the way through 2002. I cut it off there because I ran out of slide.

Q. In what ways can pirates obtain ROM contents?

A. In general there are three ways of getting ROM contents: physical extraction, theft, and glitching.

Q. Okay. Let's talk about physical extraction, or invasive attacks. How difficult are those?

A. Invasive attacks, this is the testimony you've heard about where you use a FIB or a scanning electron microscope, or so on, to get in and extract the chip's contents. So that is a moderately -- well, a reasonably difficult, fairly expensive means of extracting the ROM contents.

Q. Have you seen evidence that that has been in the public literature well before the Haifa report?

A. Yes, I have.

What I'd like to show you, ladies and gentlemen, is a paper written by Ross Anderson in 1996.

Q. Who was Ross Anderson?
A. Ross Anderson is professor of computer security engineering, or something like that, at Cambridge University in England. He is arguably the world's foremost authority on computer security.
Q. Are you in agreement with Dr. Rubin on that point?
A. Yes. Actually, I'll show you this slide. This is what I found on amazon.com where Dr. Rubin was talking about Ross Anderson's book, Security Engineering. You can see Dr. Rubin has some very nice things to say about the book. I must agree. It's a terrific book. If you have any interest at all in this topic, it's quite readable, and I quite recommend it.
Q. What did the paper that -- Mr. Anderson have to say?
A. This paper was fascinating. Okay? What we have here is the front page of the paper, and in the abstract -- you can read it all, but the key thing is Smart Cards are broken routinely. Okay? So this is Professor Anderson talking in 1996.
Q. And where was this paper presented?
A. This paper was presented at USENIX. Okay? USENIX is not some obscure body. This is the premier advanced computing society in the world. In fact, Dr. Rubin sat on the board of USENIX for a few years.
Q. Is there anything else interesting about Mr. Anderson's

1996 article?
A. Oh, many things. Let me show you this one. He goes on to say "Smart Cards are broken routinely, and even a device that was described by a government signals agency as the 'most secure processor generally available' turns out to be vulnerable."
So that is the best the government can do. The NSA and the rest of it, in '96, isn't good enough.
Q. Is there anything else about this article that you relied upon?
A. Yes. In this paper Professor Anderson described some of the ways in which you can attack Smart Cards. And here's an interesting quote. "We will now briefly describe some of the techniques available in professionally equipped semiconductor laboratories, of which there are several hundred worldwide."
So this is in 1996. Professor Anderson is saying there's hundreds of labs worldwide with sophisticated equipment capable of attacking Smart Cards.
Q. And does he discuss the ability to rent time on such equipment?
A. Yes. How's about this for an interesting quote? "We understand, for example, that production attacks carried out by some pay-TV pirates involve the use of a focused ion beam, or FIB, workstation. Low budget attackers can rent

time on them from various semiconductor companies."
So here's the thing, ladies and gentlemen. In 1996 Professor Anderson was writing that satellite TV pirates had already used a FIB to attack a satellite TV card.
Q. Does Mr. Anderson's paper spell out exactly how to use a FIB to extract ROM contents?
A. Yes, it does. The method described in the paper is very similar to what was ultimately used by Haifa.
Q. And if a chip memory aliases, would using the Anderson method to extract show to one the memory aliasing?
A. Yes. That was a rather complicated question. But what happens, if you use the technique that's described, as you are extracting the memory contents, you will inherently see this memory aliasing occurring. And you'll see it because you'll see the same values appearing time and again when you would only expect them to appear once.
Q. And did you also look at documents in this case that referenced an analysis by TNO?
A. Yes, I did. In late 1997 and early 1998, DirecTV contracted with a firm called TNO. And what they basically said to TNO was, "We want you to hack our P3 card if you can." Okay? And TNO wrote a report on their attempts.
Q. What was the upshot of that?
A. Well, here's an interesting quote from TNO. They were -- they found it possible to rent time on a FIB for

$2500 a day. Okay. Now, bear in mind, what we're talking about here, this is full commercial rates. If you happen to be a grad student at the university that has this FIB, you don't pay much to use it.

Q. Okay. Do we have a slide showing a summary of the invasive attack information?

A. Yes. So these are some of the things that I'd like you to take away regarding extracting ROM contents using invasive techniques.

Number one, reverse engineering is a routine procedure. It is done every day in industry.

Number two, the technique used by Haifa was described in Anderson's paper in '96.

Anderson has reported that pay-TV pirates had used a FIB prior to 1996 to hack satellite TV. He says there are hundreds of labs around the world capable of deploying advanced attacks against these cards.

And then in 1998 TNO could rent time on a FIB for $2500 a day. Now, this equipment is available at universities. And that's not just my opinion. The plaintiff's consultant, Ron Ereiser, also made this point. And as I mentioned, access is typically free to this equipment for grad students.

Q. Now, what is the optical technique that's referenced at No. 8?

A. Yes, sir. As well as using a FIB to extract the ROM contents, you can also use what are called optical techniques. What you do here is you literally take a very, very high resolution picture of the chip and then, by using staining techniques and taking more pictures, you can work out whether the ROM is a zero or a 1. You get all those zeros and 1's, you've got the program.

Q. Did you see any evidence that a real live hacker in this case had used optical techniques for examining the ROM contents of a NagraVision card?

A. Yes. I believe in a day or two you'll be hearing from someone called StuntGuy. StuntGuy was the biggest hacker on the scene. He wrote a document called "The StuntGuy FAQ," frequently asked questions, which literally told you everything you needed to know about how to hack an EchoStar card.

In there he describes receiving photomicrographs of ST16 chip that's at issue in this case. And indeed, he even produced pictures of them at his deposition.

Q. Now, the TNO report dealt with invasive attack on the P3 card. Would that be more difficult or less difficult than such an attack on the NagraVision card?

A. Yeah. The P3 card would have been dramatically harder for two reasons. First of all, it's a newer generation design. Obviously this is brand-new, whereas the

NagraVision system by '98 was three, four years old.

The second thing is, in the DirecTV system, the card included what is called an ASIC. ASIC stands for application-specific integrated circuit or, in the vernacular, a custom chip. Okay.

And so TNO not only had to hack the CPU, but they also had to hack the ASIC. That is a dramatically harder thing to do.

Q. Are there noninvasive ways to obtain the ROM contents?

A. Yes, there are. The most well-known method is something called glitching.

Q. Just briefly, what is glitching?

A. Glitching is when you -- I'll back up. A microprocessor is designed to operate at a certain voltage with a certain clock. It's called a clock. If you force the microprocessor to work outside the design envelope, so you set the voltage too high or too low, you set the clock too fast or too slow, then you can literally confuse the electronics or "hiccup" it. That is called glitching.

Q. Do we have an animation that shows that?

A. Yes, we do.

What I'm going to show you here is arguably the world's stupidest bank teller, but we'll go with it anyway.

If we'd start the animation, please.

The customer is asking the bank teller, "Could I have

$10, please." And the bank teller counts out the money, and they're done.

Now let's add a glitcher to the mix.

So if we would continue the animation, please.

Customer is asking for $10. Bank teller starts to count. Along comes our glitcher. He yells, "Look." The bank teller turns around and doesn't remember that they've already handed over the money. And the glitcher continues to do that and does it and does it and does it and eventually gets all the money in the bank or, in this case, all the ROM contents. Okay.

Q. How much does it cost to build a glitcher?

A. Well, you can buy commercial glitchers for about a hundred dollars. You can build your own for about a thousand.

Q. Is glitching widely used in the pirate community?

A. Yes.

Q. And did StuntGuy discuss it in his frequently asked questions?

A. Yes, he did. Not only did he discuss it, but I've seen circuit diagrams of glitches that he designed and built.

Q. Is it also possible to steal the ROM contents?

A. Yes, of course. You can steal just about anything.

Q. And do you have a slide showing the three ways of doing that?

A. Yes, I think so.

So the first method, breaking and entering, I think we all understand this is where you break into the building, burglarize it, and off you go.

The second one is much more subtle. Because a ROM is a ROM is a ROM, if you take a copy of the ROM, it's as good as the original. And furthermore, the person you've copied it from is none the wiser that it's been taken. Okay.

So anybody that's got access to the ROM, if they can copy it and walk out the door with it, then that's as good as renting a FIB and, you know, a lab and doing the rest of it. Okay.

So the question is: Who had potential access to copy? And this was a list of people that I came up with. Okay? So -- and you can read it, but essentially actually the guys designing the system, computer backup people, cleaning staff, security personnel -- security people have to be able to get into secure areas. Senior management, a lot of leaks in companies come via senior management.

Now, here's the thing. When NagraVision designed this code, they had to send it to STMicro to have it put into chips. So, then, we've got all the employees of STMicro.

And then finally a couple others. There's some good evidence to show that the code at various times was at EchoStar and also at a company called DiviCom in Sunnyvale.

So it's a pretty wide circle of people that could potentially steal this.

Q. What is a tempest attack?

A. Right.

So hopefully, if you learn nothing else today, ladies and gentlemen, you'll enjoy this one. This is a tempest attack. The way this works is that when the electrons hit your TV screen to draw the letters on it, they emit electromagnetic radiation. If you sit outside the building with a receiver like this, you can pick up those transmissions and see them on your TV screen.

Now, this was first demonstrated in 1995. At the time, I was living in the Washington, D.C. area; and I can tell you, the federal government went nuts because they suddenly realized that all these secure computers they had were vulnerable to this type of attack. Okay? So very interesting form of attack. It's there, nonetheless.

Q. Now, have you seen any evidence to suggest the ROM contents were out in the pirate community prior to December 2000?

A. Yes. There's a huge amount of evidence suggesting that the ROM contents was out in the community.

Q. What did that evidence consist of?

A. In 1999 -- I think it was September of 1999 -- there was published on an IRC channel a list of the ROM fragments from NagraVision cards. These ROM fragments were published by six different people, and they covered both ROM 2 and ROM 3. And in several cases the people publishing the fragments said, "We have it all, and to prove it, we're just giving you an excerpt."

Okay. So this is in September 1999.

Q. Did those fragments include what's called system ROM fragments?

A. Yes, they did. If you remember the animation of normal operation, you had EEPROM, user ROM, and system ROM. Well, a large number of the fragments that were published were from the system ROM.

Q. Can a buffer overflow on this card be used to obtain system ROM?

A. No, it can't. Remember our little security card, the MACM? It turns out that if you do a buffer overflow attack on this card, one of the few things you can't do is extract the system ROM. Okay? You can't use buffer overflow to extract system ROM.

So the fact that the pirates had it meant they didn't use buffer overflow. They used either an invasive technique or glitching or they stole it from someone.

Q. Now, let's go back to the famous, or infamous, StuntGuy. Did you see any evidence that StuntGuy had the full ROM images?

A. Yes, I did.
Q. And do we have a slide on that?
A. You're going to be hearing a bit about StuntGuy today.
So this is an excerpt from the StuntGuy FAQ. And one of the things StuntGuy was very kind to do was he had what's called a change log in this document. Every time he updated it, he said why he was updating it and the information that was added.
And so you can see here: "July 15th, 2000. Completed analysis of all commands based on EROM288-02 ROM dump." Well, what you need to know is a 288-02 is the official designation for a ROM 3 card. So StuntGuy's saying, "Hey, I finished analyzing all the commands."
And then below that we've got this excerpt where he says, "...the EROM guys, for providing a good environment in which to work, good information and good sounding boards. In addition, as of 25th of August, 2000, the EROM group has managed to gain full access, including back-door commands to the EchoStar 288-02 cards."
Now, let me explain to you the significance of the back door. I'll be talking about that later. Once you have access through the back door, you've got complete control of this card. Okay? Complete control.
Q. Now, was there other evidence of ROM contents that you saw as well?

A. Yes. I have a slide that shows this. Okay?
You may recognize this. This was something that was shown to Mr. Nicolas when he was testifying. And this is an e-mail sent, I believe, by Suzanne Guggenheim. And the title is essentially Publicly Available EchoStar ROM Dump and Commented Disassembly.
Q. Okay. Have you had an opportunity to look at the attachments to that e-mail?
A. Yes, I have.
Q. And do you have a slide on that?
A. Yes. There was a readme file in here, and there was a couple of things I thought you should see.
So this first one, it says, "This file contains all of the ROM dumps of the EchoStar 288-01 cards that have been available on the Net, as well as some ROM information we got from other sources." So a 288-01, that's the official designation for a ROM 2 card.
Q. And how similar is the ROM 2 code to the ROM 3 code?
A. Oh, it's very similar. Essentially the ROM 3, they just took the ROM 2 and fixed all the bugs in it and issued it as ROM 3. So essentially the same card.
Q. And do we have another slide?
A. Yes. They go on to say why they're doing this. And here's the quote at the end: "Until eventually a working hack emerges at the far end of all of this."

Q. Is there another slide?
A. Yes. And this is where they're discussing who they are. And a couple of things I'd like you to see. "This information wasn't all discovered by just one person."
And then at the bottom there, "We and others have put a lot of time into this." So you go through this readme file, and it is very clear that there are lots of people with this ROM working on the problem of hacking the system.
Q. Now, was there something else in the zip file other than the readme file?
A. Well, absolutely. What was in there was a commented disassembly of a NagraVision ROM 2 card, including system ROM, user ROM, and EEPROM.
Q. Can system ROM be obtained using any buffer overflow attack?
A. No, absolutely not.
Q. And what is a commented disassembly of ROM?
A. Yes. So remember when I put up David Mordinson's code side by side with the Nipper code? The Nipper code you just saw was a binary image. And on the right we had David Mordinson's code, which was very nicely formatted and had all sorts of comments and explanations.
So the process of disassembly is taking the binary image, just those numbers, and back-converting it into a meaningful program that a human can read. Significant

undertaking.

MR. STONE: Your Honor, I'm going to shift to another topic. I don't know if this would be a good time.

THE COURT: This is a good time.

Ladies and gentlemen, why don't we resume at 1:00 o'clock. You're admonished not to discuss this matter amongst yourselves nor to form or express any opinion concerning this case.

Sir, why don't you step down.

THE WITNESS: Thank you.

THE COURT: All right. Counsel, have a nice lunch.

(Lunch recess held at 11:56 a.m.)

(Further proceedings reported by Jane Rule in Volume III.)

-oOo-

-oOo-

CERTIFICATE

I hereby certify that pursuant to Section 753, Title 28, United States Code, the foregoing is a true and correct transcript of the stenographically reported proceedings held in the above-entitled matter and that the transcript page format is in conformance with the regulations of the Judicial Conference of the United States.

Date: Month Date , Year

____

DEBBIE GALE, U.S. COURT REPORTER

CSR NO. 9472, RPR