EXHIBIT J

-TRANSCRIPT-

27th Chaos Communication Congress

"Console Hacking 2010, PS3 Epic Fail"

December 29, 2010

Male 1: Just a quick survey, who of you owns a Wii? Please put up your hands. Who of you owns an Xbox? So, for you, there's a book out there by [Ronnie ?]. Who of you owns a PS3? Woah! That's a majority. So, with all these consoles, which means a lot of computing power at our hands means a lot of stuff to do, what we want, at least to my definition of ownership and properties, so how to peek into the PS3 and consoles in general, here is a Fail Overflow.

Bushing: Alright. Great. Thank you. So speaking of owning a PS3, actually, we'll get to that in a minute. I should start out by saying that I am Bushing and here we have Sven, Marcan and Sahar. We are… Sahar, excuse me. We are just four members of Team Fail Overflow. Unfortunately we sort of overflowed the number of speakers so the rest of them are, you know, here in the front aisle. It's an international team with, I think we have members on almost every continent. Let's see, we may be familiar faces because two years ago, Marcan and I presented on some of the hacks we'd done for the Wii. A year before that, I had demonstrated the first the unsigned code execution on the Wii here at 24C3. 2008 was the full reconstruction of the [intentional jibber by speaker] to be completely broken. That was a fun couple of weeks. So, on the subject of the Wii, since that's really what we have the most experience with or are most known for. Wii's been out for about three years and three years since we demonstrated that unsigned code execution. In that time, Nintendo has released nine firmware updates, and of all those updates, eight of them were contrived. Only one of them had an actual real feature that somebody would want; namely the ability to run games of off an SD card. Every single other feature was background improvements to improve system performance which really was their code for patching vulnerabilities. According

to VG charts, they sold about 73 million consoles which is the highest of all the consoles. Correct? And of those, we estimate that about 30 million of them were produced before they fixed the bug, where the hash comparison bug that we had disclosed. And so, these people were fortunate enough to be about to run, "bootme," our alternate bootloader that we wrote for the Wii. We actually ended up putting a lot of time into that because one of the biggest challenges for us hacking on the Wii was not really the protection, but really the fragility of the thing. The software is very, very fragile and there are very many ways to accidentally break the device and no way to fix it. So we put time into making something that would simply let you unbreak your Wii. And since September of this year, we are proud to announced we have 1 million unique users of the Homebrew Channel. That's over 1% of all users, all owners Wiis, have installed our Homebrew Channel. So, I say we had a pretty good run.

So, let's take a look at what consoles we have here. So, there is first the Wii which was released at the end of 2006. Then the Xbox 360 which was released about a year earlier and the PS3 which was released about the same time like the Xbox and like the Wii. And so the Wii was pretty much broken from the beginning because there were like Drive chips which allows pirated games to be run Game Homebreak and then after a year it was like fully broken and Nintendo tried to fix it again and again but, well, they failed. So, in the end it's just broken, and if it's still is broken and it will properly be broken. So, then we have the Xbox 360. It has a really good security system and there were only like two major hacks which was the King Kong hack which was demonstrated I think two years ago or three years. And then the JTAG hack, but those were minor bugs and Microsoft was just able to fix them. So, okay, the drives are completely hacked so we could also mark the whole thing here in red as a block but, well, we don't care about piracy. It doesn't allow homebrew codes and, we don't care about running code others wrote. We want to run our own code. So, and there is finally the PS3. So, the

difference here is that the PS3 originates a part of Linux. So, no one really cared other than one single thing where you cannot really use these 3D feature on Linux, so they found some way to get around it. Sony pushed a firm update because they want that and to fix it, so well, but still nothing happened. But then Sony decided to release the Slim so they claim they don't want Linux or they're not going to support Linux because they want to save money. Well, today we know that this is bullshit because then it just runs on the Slim without any huge code modifications. So it was just something they claim to do, but that is certainly not true. And after that, people actually started to care and a few months later, Geohot tried to break the hypervisor which is going to be explained later. Sony's reaction was to remove Linux and then finally Jailbreak present some new hacks and the Play Station 3 is likely going to look like the Wii afterwards.

Steil: Hi. My name is Michael Steil. I'm not actually part of those guys, but if you remember, there's been a console hacking talk for at least the last eight years, every year. Every once in a while, I get a cameo in one of these talks because I look at big picture at the statistics about what is going on in the hacking space. And it's always the same thing I want to say which is Linux is inevitable, and either you support Linux on your hardware or it will be hacked so it will be run sooner or later.

Correlation is not causation, but I have some good statistics to back that up. So, if you look at the devices list, this is a slide that I presented three years ago in another one of these talks. These were the systems that were available at that time, and the years when they came out and how good the security system was. So, darker, darker red colors means very good, very sophisticated, very complicated security system and as time progressed, it got better but smaller systems still had weaker security systems because it just wasn't that easy to implement them in smaller space. This column shows you how long it took to have these systems hacked and there are some correlation between a complicated security system and

hacking the system. And the question is why did it get hacked? And the statistics clearly show here, that except for the first one, the PlayStation, very long time ago. All of them were always done for the sole purpose of running Homebrew software, running Linux, just running your own code because you owned the hardware. And there has always been that side effect of well if you open it, if you can run anything on it, you can run pirated games on it because the keys have leaked or the whole DRN system is just not in place anymore. But there is one here that is special. No, wait! Let me first update the slide. So, the Ipad, same thing as the Iphone. Okay, so let's look at the PlayStation 3. This one is special because while the security system looks like it's kind of sophisticated and as of, well, three years ago, or even a year ago, it was not hacked. So, this has changed. So, now after four years, it has finally been hacked, and you could say it got hacked for a piracy Homebrew. I let those guys decide and argue on that. And of course there is always the side effects, so piracy is possible or was possible, whatever. But you could also look at it from another perspective , which is if you just assume that PlayStation that we had in that list was the old PlayStation where Linux was possible and that was the whole reason as I argue that it never got hacked, it ran Linux already, it didn't get hacked, never. So, now lets update this for how long did it take from the point when the system was closed until it got hacked. And then we have just 12 months here which is pretty much the same as the Xbox 360 with a similar security system anyway. Okay. Thanks.

Male 2: Okay. So, before we're going to talk about how we can break the security system, we have to talk about how the PS3 actually works. So, what Sony did was to ask IBM for their cell processor which is called the cell Broadband engine as they put it in there. So, this is essentially just a 60 fobit (?) and they have like, which is on the bottom left there, and they have like eight SPEs or SPUs. That's not a logistic processing element. It's just a fancy name for a backed-up processor. So each of those things have like 265 KB of memory. While, it cannot access the rest of the

memory. It can just be emanating from and to it. So they can use this for security because there is this special note. At the bottom, you see the local storage? You really, you have like the whole thing is available from the Power PC. You can just read it. You can write to it. You can even stop the SPUs and look at what they are doing. You can even write a debugger on the Power PC to like totally inspect what's in every single register. However when isolation mode is enabled, an authenticated binary is loaded, decrypted and verified by hardware, the local storage gets closed down, all the debugger features are disabled so the only thing you see is the part marked in green there, which is like a small area used for communication with the Power PC and the red area is blocked and only accessible from the SUSL. So Sony can use this to like hide things or implement some stuff they don't want us to see. So then, we have how codes are running in there. We have the Hypervisor at the top which can virtualize multiple logic partitions and usually the only thing running there is Level 2 or GameOS which is like some kind of kernal, it provides support features for games and so on. And games itself are running on Problem State which is the Power PC. And Level 2 and Level 1 can communicate with the SPU. What usually happens, or what Sony does is that GameOS cell Level 1 with some hyper calls tells it to like, load SPU, load some loader on there and let it encrypt up. But GameOS can do this as well.

So, and now we going to talk a bit about the boot process. Okay. That's actually wrong. It's actually Bootloader here. Well, so, Bootloader essentially loads Level 0 which is some Power PC code running in Hypervisor mode. The sole purpose of this code to bring up the Power PC and bring up another SPU to load metldr which then loads Level 1 loader so those are just isolated scenarios used for like loading all stuff. And finally, Level 1 gets loaded here. It just continues with loading another SPU loading metldr loader again and loading Level 2 loader, so what metldr it decrypts Level 2 loader then Level 2 loader decrypts finally Level 2

which is then running in kernal mode on the PS3. So, red is PE and blue is Power PC. So, two CPUs kind of interlead the code.

Male 3: Alright. So, let's look at the security systems of the different kind of consoles. There is a whole bunch of features that consoles can share in their security systems, so lets compare them. The first security feature that everyone puts on is the hidden bootRAM that can do something special. It was implemented in the Xbox 1 in the southbridge chip and every console after that has put it in the CPU which is very nice 'cause you can't easily read it and obviously you can't overwrite it with a mod chip, so you pretty much need that. After that, of course people use public encryptographers to authenticate the code that runs on consoles so both the Wii and the 360 have that, and why am I talking about that. So all of the four consoles have that and the Wii and the 360 also have On-diestorage which hides keys so that you can't easily read them from external memory or anything like that. The PS3 kind of has that but it doesn't work very well. Since we have public Public-Key crypto, of course you want to keep that chain of signed and verified code running from boot so you need a chain of trust. The Xbox 360 and the PS3 do that. The Wii tries to do that, but they only verify install code and install time, so that kind of breaks the chain of trust. That's why it doesn't get that point there. The Wii, the 360 and the PS3 also have per console keys. The Xbox 1 did not have that, so these keys can be used to encrypt things like internal code and tech thing like that to be unique to one console. The Xbox 1, the 360 and the PS3 have signed executables. The Wii doesn't sign executables. It kind of has packages. So, it's an outer layer. With signed executables of course you ensure that any code that you run on the system is authenticated before running it. Okay. The Wii and the PS3 also have a co-processor. You saw the Security SPon the PS3, and the Wii has what we call the Starlette, which also runs all the security code. So, this helps isolate the security subsystem from the rest of the code and, you know, prevent exploits from sort of leaking into the security stuff. The Wii

does something that's really interesting, which is it fully encrypts and signs an entire disc image so you can't touch any single resource file in a game. It uses a hash tree which is a little clever construction and it effectively signs and verifies the entire thing as you read it from the disc, so you can't change anything. The Wii and the PS3 also have encrypted storage. The hard drive on the PS3 and the flash memory on the Wii are both encrypted with a per console key, and so that way you can't read it easily from the outside and figure out what's going on inside the internal storage memory of the consoles. The Wii, however, also signs that code with an HMA signature so that you can't patch stuff into that storage even if you can decrypt it somehow without knowing the key that is particular to that console that lets you do that. So that's quite interesting too. Quite important, too, if you encrypted storage.

The 360 does memory encryption which none of the other consoles do, and it also does memory caching, so you can't tap a memory buzz or glitch something, in the protective areas of memory. You can't DMA to them. It basically prevents a whole bunch of hardware based and DMA based memory attacks.

And the 360 and the PS3 have a hypervisor which helps supervise other application code running on them, and does some of the security stuff in the hypervisor. The PS3, however, also has user and kernal mode, so the PS3 has all the three levels; user, kernel and hypervisor mode while the 360 only has kernel and hypervisor mode it does not do user mode. It runs all games in kernel mode. And Wii runs everything in kernel mode. It doesn't even try to separate stuff on the main CPU. And the 360 has e-fuses, which are used to prevent downgrades. It actually blows a fuse inside the CPU when you perform some system updates and you can't reverse that any way, so once you run an update there is no way you can downgrade unless you can break the chain of trust before that check happens which, so far as I know, has not happened so far. So these are the features of the consoles. There is one, however, on the PS3 that is already kind of useless

because as it turns out the PS3 encrypts stuff with a key that is the same for every sector and the same IV and doesn't hache anything. So, they pulled of this neat trick where you can copy a file into your PS3, like a movie or something, then you can find those sectors by watching what changed on the drive. Then you take another any chunk of the drive, copy it on top then read the movie back and it decrypts those sectors that you copied on top for you. So, you can do that to decrypt any chunk of the hard drive and since they don't verify anything and everything uses the same key and the same IV, it just works. So that encrypted storage is pretty much screwed up to begin with. Okay, so, the PS3, however, has OtherOS, which is a way to run Linux and as you, which we saw, well, I'm pretty sure it worked as a deterrent because people could already run codes so why break it? But then Sony decided to get rid of it on the PS3 Slim which is, you know, a way of drawing attention.

Now people are going to start looking at your system, and we know now that there is no technical reason not to have Linux on the PS3 Slim. So, some marketing or some kind of, you know, some boss somewhere said that we don't want Linux or we don't want people running their own code or we don't know why. But, they definitely drew people's attention with this, and now people started trying to hack into the PS3, and Geohot tried to give it a shot. So, you may know that on the Wii, our first exploit was a Twiizer Attack which worked by shorting out some RAM address lines and glitches the RAM bus to access areas of memory that we were not supposed to access. Geohot did something like that on the PS3. He glitched the RAM bus to essentially access the areas you're not supposed to access. So, the way this works is usually is you have a hypervisor and a kernel on the same chunk of memory, but the hypervisor controls the page tables to control memory access permissions for the kernel. So when allocate some memory from the kernel, the hypervisor gives you a mapping to that memory so that you can access it and when you remove that chunk of memory it deletes that mapping and you cannot access it

anymore. Geohot glitched the memory bus right when that happened so that even though a hypervisor thinks you have that chunk of memory allocated, in reality it's not actually, it's not free completely because the kernel has still a page entry that can access it. So the hypervisor thinks it's free memory, but even still read and write from it which is bad. And finally, all you have to do is ask the hypervisor to create a new virtual address space which means create a new page table, and if you're lucky, it ends up in that chunk of RAM and then suddenly you have access to the page table. You can read and write from it. If you can write to the page table, you can map the hypervisor. If you can map the hypervisor, you can do anything can you want in hypervisor mode. So that gives us hypervisor exposure. So this was kind of an academic hack. It works and you can do stuff in hypervisor mode, but it required really annoying hardware to pull off. It was really unreliable, and even when you got hypervisor dumped, no one really knew what to do with them. They were kind of there and, whatever. Sony, for some reason, got really, really, really annoyed at this because they decided to piss a lot of people off by removing OtherOS completely from old PS3s. I'm pretty sure that violated some European consumer protection laws and stuff. So, but the worst part of this is that, the people who used OtherOS are the hackers so by doing this, Sony pissed off the hackers. That's a really, really bad idea. In other words, they are so getting hacked now. So, for a while, interestingly, nothing really happened and obviously someone was working on this behind the scenes because then we got the PS jail break and then we got a whole ton of clones of it. So the PS Jail Break is a device that lets you run your own code at LV 2 kernel level and above and it uses a clever USB exploit to do that. It's actually a USB device that pretends to be multiple USB devices behind the hub. We'll ignore most of those for now because we gotta keep this short. But, the important ones are the first one and fourth owner's devices as we call them. The first one just delivers a payload that is part of a USB information descriptor into the LV2 kernel It's sole purpose is to put the Payload

into memory. It didn't really do anything once it's there. And device number 4 gets loaded and here's where the magic happens. There's some stuff that happened before that. So, device number 4 has several configuration descriptors. The first one is loaded from the device, and that works out fine. It gets put into this blue buffer and the second descriptor gets loaded, but something weird happens. The PS Jailbreak does an interesting glitch with USB. When you read a descriptor it has a total length but to read the descriptor, you need the length so it's a chicken and egg problem. USB solved this by reading the first eight or so bytes to begin with, then you read the length from that, then you read the whole descriptor again once you know the actual length and you can re-install data. So, the PS Jailbreak returns a normal descriptor link the first time it around when it reads the first eight bytes. Then the second time around, it actually returns zero. That forces the USB code to glitch and never actually copy the descriptor out 'cause now it thinks its length is zero even though it read it. So, the funny thing is, well, LV2 is gonna try to patch that descriptor that's not actually there. So it's uninitialized memory. What's actually there is, it turns out that device number two was plugged in before and then unplugged and it sort of initialized that memory for LV2, and it has these four funny bytes at the bottom. O421B42F. When you patch that as a configuration descriptor, which is what its gonna try to do, it thinks it's a descriptor with length 2FB4 which is really, really long. It's past the buffer. So we've overflowed this buffer it and now it tries to load configuration number three and it jumps 2FB4 bytes forward out the buffer and into a C++ object array that actually belongs to device number three. Yeah, three. So, it puts this configuration descriptor right on top of C++ objects, and it overwrites the V-table of C++ object to .2 the payload. This V-table holds function pointers for things like the destructors and some virtual methods and things like that, so when these objects get destroyed they actually run our own code. So, when device number

three which has a bunch of crap that created these objects gets unplugged, we gain LV2 code execution.

So this was implemented by the PS Jailbreak and then cloned and there are a whole bunch of implementations of it. The funny thing is, okay, so, this works. Why does it work? Why can't we just run code from data memory. This is to solve the problem, right? Well, it turns out that LV2 does not do writes or execute on its kernel. It does not try to protect executable, I mean, data from being executed. Which is kind of silly, 'cause, you know, we've had this for a while now. But even more importantly, the hypervisor doesn't know anything about this because the hypervisor will happily map any memory you want that is executable. Unlike on the 360 where the hypervisor actually verifies anything mapped as executable, so it can guarantee that any code running has been signed. On the PS3, hypervisor doesn't even try to do that. It's a hypervisor meant for virtualizing operating systems. Not for security. So it's the wrong kind of hypervisor. It really doesn't do what you want it to do for a security system. It just kind of sits there and looks nice and doesn't protect you from these kinds of bugs. So, okay, so we have LV2 compromised. We have not compromised the hypervisor and we have not compromised the Security SPE, so all those are fine. So what happens now? Well. So, why on Earth can we pirate games by just compromising LV2? Well it's because the security system makes no sense. So it turns out that you can just copy games to the hard drive, patch LV2 to run them from the hard drive and LV1 doesn't care, and the Security SPE doesn't care so you can break 20% of the security and copy games which is 100% of what Sony doesn't want you to do.

So let's go back to our overview of the security features. Obviously, the hypervisor is basically useless because it doesn't prevent you from copying games, so what on Earth is it doing? And it's not preventing you from running your own code either 'cause you can just use an exploit. And the signed executables are also

pretty useless because the hypervisor does not enforce them to be signed, to be loaded into memory. You can just ask it for some Executable pages and it just does that. Okay, so, Sony fixed this, obviously, and when Sony fixes something, as you might know from the PS3 Slim, everyone thinks, "Okay, let's downgrade." So people started looking into this and I believe it was the people behind the PS Jailbreak, who also came up with a way of downgrading and the way this works is that Sony has a service mode that they use in their service centers that puts the PS3 into a special mode where it can run some sign code from a USB stick. This service mode is entered by using a USB dongle that does some down code with the PS3, but it turns out it's a symmetric off its HMAC and since people had broken into the PS3, they found the keys, dumped them, and made their own clone jig which is what they call them. And there was a service app that was signed, that was leaked, that lets you re-install the operating system on a PS3 without any kind of downgrade check. So, you just use this jig then put this on a USB stick, put a system install file on a USB drive, and it just installs that without checking if its older or newer or anything like that. And since people can downgrade, back to piracy.

Okay, so what we did with this whole PS Jailbreak thing, is we wrote a Asbestos which is a replacement for GameOS. OtherOS. The idea is that, well, since GameOS and OtherOS really are kind of the same thing with different permissions, the hypervisor interface is the same. You can basically replace level two with Linux. So, Asbestos takes over Level 2 and then you can pretty much just run a Linux kernel, bootload it from a network or something like that, and you can run Linux again, even on a PS3 Slim. It turned out it just worked. There's nothing about the Slim that makes Linux not work. So it was all, you know, the reason why it doesn't support it is because it didn't want to support it. And we also added feature to it that lets you make your PS3 into a dumb slave that you can control from your PC and you can use this to experiment with python scripts,

poking the hypervisor, poking the SPE's, rewriting memory, making hypercalls, basically it's a way to experiment with PS3, and especially its security system, really, really quickly from a PC environment.

Male 2: Okay?

Male 3: Yeah.

Male 2: So, so what have done so far? So far we managed to break into Level 2 with some jig where we have plug something into USB bus[?], then turn it on, press eject, and then it actually runs your code. But, we want more than that. We want like, some, vulnerability where we can just turn the PS3 on and just boot into Linux for us. So, to do that, we need to figure out more stuff like, how does Sony encrypt their EFLs? So, what they do is they built a normal ELF using just some compiler and stick a header on top of that. So, this is their encrpto in blue and green here. They have each of those loaders running under isolated SPUs loader[?] key which can be used to decrypt a unique SELF key. This one is then used to decrypt the rest of the header, and at this point, the signature is tricked. So, the signature we're going to talk about that later. It's essentially all the stuff marked in blue is signed. So you can't modify anything in there. And then the header also has a table of AES keys and SHA-1 so that they can actually encrypt the program headers and make sure that you don't modify them. So, as soon as you modify a bit somewhere in the whole SELF it will not work anymore because either some hashes not, that doesn't verify anymore or because the signature is well, its wrong. So, like every code Sony gives you is wrapped into those files so you can't see anything. However, we can just use the SPEs to decrypt the code. So, Sony's idea is, yeah, they can't see our code so they can't find box in there. Well, as soon as we have power pre access, we can just ask the SPU to nicely decrypt stuff for us and it will do so. It does not do any checks on what's actually asked you to decrypt things. So, with this, we just use Asbestos and this net RPC thing, write some python scripts on the PC, and use it to like decrypt application[?] cells, or Level 2, or like,

decrypt just about everything now. They're just asked you to do it. So we don't even need to know how it works. We just have to ask it to do it for us. So, yeah, the obfuscation is useless, but, we still want keys. But let's first take a look at our table again, what we did now. So we have the security code processor there, its pointless. So, let's go one step back the chain of trust I showed at the beginning. So, here we have it again, and I'm not going to explain the, so, and on the left you just see what is run in chronological order, and on the right you see the usage. And there are two things there which they can't update. It's boot loader and metldr loader. So, boot loader is the very first thing that runs and just, yeah, it's just the very first thing that runs. And a metldr loader is used to load isolated SBU binaries so they can't update that because its encrypted with the console specific key. The rest can all, they can update all the other things. And they don't want us to downgrade. So they added some revocation. That you get revocation list as well and the loader verify if you've tried to load some binary, "Is this on the revocation list?" and if it is, and if it is, it will just refuse to boot. So, the stuff at the top, it cannot be revoked because the previous loaders do not support any kind of revocation lists. So Sony did something else there. First it reads boot loader LV0, metldr, LV1 loader, and LV1 from the Nert or from the noman PS3 and just runs them. And then once everyone runs, it loads all that stuff again to verify it. So you can just remove the hack in there, so when its read first, you supply your downgraded version. Then when the PS3 reads it for the second time, you supply the real version its supposed to run, and it will just run because it thinks like, yeah, I am running the current version so I can just continue to boot. And this allows us to like, downgrade everything. Because we can just downgrade the revocation list as well, so the revocation is useless. It's just some kind of specification thing. There's no revocation there. So as soon as we break one loader, we break it forever. And we can forever own all PS3's out there. So we need to do that.

So, here we have the local SP ELF storage again. It's in isolated mode so you can't access the red stuff. Only the SPU cells can do that. But you can access the green stuff, and you need to supply a revocation list. So this list is, of course, signed and encrypted, so they have to copy it to the protected part. Because they don't want you, if they were to decrypted in place and checked the signature in place, you could just like wait for it to do that and then at the right time, write some Power PC code to modify it. So, they really have to copy it, and here's how they do that.

So, what's wrong here? The problem is that we have the destination buffer live before the LV2 loader code. So we control the source and we control the length of the source. So what if we just write some really, really huge value there? Well, we overwrite LV2 loader code and run code in isolated SVU mode, then we can just go ahead and dump their keys, just write something to push them out to the Power PC and then, yeah. Then we can actually revoke the whole loader and we put some keys there.

So, what are the implications of this? So, it's only a bug in one isolated loader. However, if we put such a modified revocation list onto the nor flesh(?) will just load it. It does not care about the size and that code actually works, and it will just supply a really huge revocation list to like LV2 loader. Then we patch LV2 loader in our signatures, and well, we get our code running at boot time without any kind of dongles or stuff. And this breaks the chain of trust pretty early.

So as you can see there, this has failed. But we promised epic fail so we'll have to look further.

So, let's go back to the table, well, the chain of trust is broken. And back to the cells. We now like how just about everything works here except for the signature pod. So, how do they do that? How does the signature thing work? So, this is ECDSA some crazy math stuff that guy over there is going to explain now.

Male 4: Hi. Okay, so the thing they're trying to do is figure out the product key they used to put signature on files. You normally cannot do that, of course. That's why it's a good breaker system. As they say there's a lot of parameters that are public, BABGNQERS, whatever. But there are two things that are private in the single signature, that's K, the private key, which we want to have. And it's M, but it's supposed to be a random number. Next slide. Okay, so how does someone who signs something calculate the number's R and S inside the signature? Well, R is done by a scale of multiplication of base point of the elliptic curve difficult stuff. It's a difficult problem that's the base of the security of elliptic curve. But S is just calculated as normal numbers. That first equation, we cannot solve. No one in the world can solve it, so lets just ignore it. The second equation, it's got two unknowns in it now, okay? And so we cannot solve for it either. But M is supposed to be a random number, and for some reason, Sony uses the same random number all the time. So, that solves our two equations again, but twice now like you have two separate signatures. We're going to ignore that tophonic again because it is difficult, right? But now the equation for S1 and S2, we still have only two unknowns because it's the same, M. It's supposed to be different M every time. Then we've got three unknowns, only two equations and the way to solve it. But now its got only two unknowns, so it is trivial to this. So, for M, you take the two formulas, effect them, whatever, just a bit of formula manipulation and then you solve M, and you've got M so you just fill it and you solve for K. So we've got the five key without even having to know most of the encrypted entry.

Male 1: So we actually used ECDSA in the Homebrew Channels network update functions so that someone can't own your Wii through a man in the middle attack or something. So, this is how we do it right? We do the easy math. You can see the see the end terms ECDSA blah blah stuff there, and for M, we read cryptic completely secure random numbers from the brand name Linux. You know, that is what you are supposed to and what Sony does as well.

And of course, as Sahar explained, if you use the same M you can calculate K once you have two signatures. If you have K, well that is a private key. And with a private key, you can sign things.

So, these signatures are every bit as valid as Sony's official signatures. They are indistinguishable and we can get keys for LV2. We can get keys for LV1. We can get keys for revocation list. We can get keys for high professional configuration files, which is interesting, and for packages and a whole bunch of stuff. So…

Male 4: We're actually don't have keys for Level 1 because we can't run that loader because are topped with that stuff, so we still have to figure that out.

Male 1: Yeah, sorry. So, in fact you can get the keys if you have the plain text the loaders, but right now Sony's security for the few loaders that haven't been dumped yet it hinges on just the AS stuff that we don't have, and the per console key. Everything that you need to get these keys is inside the PS3 you just have to get to it.

So, back to the table here. Well, they botched the public key crypto so that's epic fail. And were left with user kernel mode and on-die bootROM which are not exactly high-tech security features, so, pretty much botched the entire thing. Thanks Sony! Alright.

Bushing: Looks like we might have a little bit of time for questions? For Q&A?

Male 1: We were actually going to try to do a demo of this, and we were going to do it now, but as usual, demos are late or don't work. In fact, we have video issues the same way. We have video issues with the BootMe demo two years ago, so we're going to try to fix that, and we're trying to schedule a lightning talk tomorrow to get that presented.

Male 1 So, if you have any questions, we have two microphones. On the left. On the right. Please line up. And if you're afraid of asking questions in public, the people are down at the x center.

Member 1: I got a question. Do you know where the PS Jailbreak came from?

Bushing: Not a clue. Seriously. Not a clue. We can guess, but we don't have any proof for absolutely anything.

Male 5: We think it's somewhere from the southern hemisphere.

Member 1: Right now, I noticed that the, not only the exploits, but the USB stack exploit but also the payloads has been completely reverse engineered, so there are source code available right now? Everything?

Male 1: Yeah, well, right now, Asbestos is public. It has been for a while. It contains an implementation of the USB exploit that runs on the TIO ver processors. There are a whole bunch of clones for other chips, and of course, there's also a reverse engineered payload for that if you want to replicate the original PS Jailbreak functionality instead of running Linux. That's out there right now. Our stuff, we, well, we did some pretty terrible hacks to make it work for the demo. Hopefully tomorrow, so we need to clean things up and make it saner, but, I mean, expect this in the next month or something like that? We'll release clean tools for the stuff that we did here.

Member 1: Thank you.

Member 2: Okay. Question from the DRFC. Now that you can sign your own code, why can't you create a Blu-Ray software payload?

Bushing: Could you repeat that?

Member 2: Now that you can sign your own code, why can't you create a Blu-Ray software payload?

Male 1: So, we actually don't have the key to sign games because that's App Loader, and they changed those keys several times. Actually, no, they didn't change those keys. Why don't we have that? We do have an exploit in App Loader. Sorry. So, yeah, you need the plain text of a overloader to get the public key. And once you have the public key, then that. So, we can't sign games, and also, we don't really care about games because we care about global (?) access.

Bushing: It's probably possible, eventually, but we're not planning on working on it.

So, anymore questions? No? So, if you'll leave the room, please go to the right. Pick all the trash which is around you and everything that's valuable, please put in the lost and found. Thank you.