EPISODE 443

[INTRODUCTION]

[0:00:01.1] JM:

[SPONSOR MESSAGE]

[0:00:13.0] JM: You are programming a new service for your users, or you are hacking on a side project. Whatever you're building, you need to send e-mail. For sending e-mail, developers use SendGrid. Send Grid is the API for e-mail trusted by developers.

Send transactional e-mails through the SendGrid API. Build marketing campaigns with a beautiful interface for crafting the perfect e-mail. SendGrid is trusted by Uber, Airbnb and Spotify. But anyone can start for free and send 40,000 e-mails in their first month. After the first month, you can send 100 e-mails per day for free.

Just go to sendgrid.com/sedaily to get started. Your e-mail is important. Make sure it gets delivered properly with SendGrid, a leading e-mail platform. Get started with 40,000 e-mails your first month at sendgrid.com/sedaily. That's sendgrid.com/sedaily.

[INTERVIEW]

[0:01:26.6] Host: Emin Gun Sirer is an associate professor at Cornell University, where he's the co-director of the initiative for cryptocurrencies and contracts. His research spans cryptocurrency's networking, operating systems and distributed systems. Professor Sirer is one of the foremost researchers and bloggers in the space of cryptocurrencies and smart contracts. Today he'll be chatting with us about smart contract security. Gun, thanks for coming on the show with us.

[0:01:47.9] EGS: Thank you very much for having me.

[0:01:50.1] Host: Cool. Smart contract security, I want to start with the basics. Our listeners are going to be familiar with the cause of the blockchains, we'll assume they're also familiar with Bitcoin. Describe for me what exactly is a smart contract.

[0:02:02.5] EGS: A smart contract at its core is a program that can programmatically manage money flows or asset flows. Essentially, the way I think of this is for the longest time we had programs that manipulated most pixels on the screen. But you type into them and they take input from you, or they take input from files and they produce things that are either other files or perhaps an output for you to look at.

That was the dominant paradigm for computer programming for the first few decades of computing. Sure we always had actuators and so forth and sensors as well that was a smaller niche, especially the field of robotics has been taking off on that front. That's its own separate thing, but when it came to general purpose computing, literally the programs were limited to pixel manipulation and/or bit manipulation.

Now with the advent of blockchains coupled into these programs into smart contracts, you can actually have programs that programmatically direct assets, whether they're money or tokens or tokens that you will present real-world objects, whatever they might be, they can direct those resources according to a preset algorithm.

You and I and other people can agree what that's algorithm ought to be for all time. Then we carry through with the smart contract knowing in full confidence that the program can't only do those things that it has been programmed to do. That's an amazing, amazing ability. It seems like a smallish thing, but it's not, because the entire notion of counter-party risk or trustee risk has been taken out of the equation.

There is no fear that our agent who is acting on our behalf who we would normally have to tap into for directing money flows is going to misbehave, because our agent is a program. It cannot misbehave. It can only do that which it has been programmed to do.

This I think is an amazing new idea. If I look backwards at the history of mankind, if I look what we've done overall in terms of organization, social organization, there are only a few things that

stick out. Starting with savage, living out in the woods, hunter gathering society. Until now, if you think of other big events there, written laws, that's really enabled a new level of civilization.

Corporations that shield their participants from personal liability, that opened up another way of innovation. If I think about the gap between the east and the west, a lot of it is traceable not to actual technical things, but to social organizational advancements of this kind. Smart contracts I see as being one of these things that really open up and that new set of possibilities.

[0:04:45.2] Host: Right. To further motivate that, can you give me just some examples of maybe low-hanging fruit of things that smart contracts entering into the normal world of commerce, what could they immediately solve or make much simpler.

[0:04:57.3] EGS: Sure. There are many simple ones. The simplest one that if I were to look at my newsfeed, just about every two weeks or so, I hear of some musicians suing their agent, because they did not get their share of the royalties. This I think is one of the simplest uses of contract, where of the money coming in, some person needs to go in one direction and another person at least needs to go into another direction.

The middle man there ends up – if you trust him or her with your money flows, then they can misbehave. A smart contract can take out the unknown ability, the unknownness of that person. So that's the easiest one. But we get into much more exciting ones with insurance for example. If I think about an insurance company, it's really a peer-to-peer company, right?

I pay into the system. So do you. On occasion something bad happens to someone and someone predictable flashing and then the system are to pay out. The intermediaries in that system, they are not bringing the value. The value is coming in from the network of people. If you could actually create the same kind of network without the intermediaries, you could actually affect an enormous efficiency increase.

[0:06:08.5] Host: Excellent. We're all here a technical audience and we know that blockchains or public blockchains operate over peer-to-peer network. What exactly happens in this peer-to-peer network when a smart contract gets executed? Describe for me what all the computers and the network actually do, what' s the consensus process, how does this happen?

[0:06:25.3] EGS: Sure. At its core, a smart contract platform, like Ethereum, Tezos and the other contenders in that space, they execute a unified giant state machine. That's a machine that takes steps in tandem. There are many, many replicas of it, and all the replicas take the same step at the same time step.

The way they operate is we all decide, all of the machines decide that on the next sequence of steps to take. So they decide for example, okay Professor Sirer is playing chess and he is moving his pawn from E2 TO E4 or something like that. Hasiba is writing, underwriting an insurance contract. He wants to put up, I don't know a $100,000. And wants to ensure somebody in Florida against let's say. Let's suppose that you have some inside information on weather patterns they see.

These are the next few steps that the machines have to take. There is an initial consensus protocol. This is where a lot of the magic has to happen, where we all have to agree on the same sequence of actions that are to take place. Then we execute those sequence of actions according to their associated smart contract code.
As we do so, we update the state machine and the state machine goes to its next step, whatever it may be. It essentially says things like, "Okay, well Hasib having underwritten this contract matches with this particular farmer or whoever else," and Professor Sirer having done E2-E4 is facing the opponent's move.

[0:08:03.7] Host: To put it simply then, smart contract is just a series of state transitions that all of these replicated state machines inside of a smart contract platform are executing in the same work?

[0:08:16.4] EGS: Pretty much. Yeah, a smart contract is a definition of – Yes, it's a series of commands that you can execute and their associated effects. What are the acceptable moves you can make at any one time with regard to a particular arrangement?

So in the chess game, for example, I only have so many moves. The rules of chess are encoded in the smart contract. The insurance contract, you'll have a matching engine and so forth, and the conditions under which you match. Then the underlying platform decides on what

the inputs are from the users, sequences them using this process of consensus, and then executes them.

[0:08:59.5] Host: Bitcoin, which most of our listeners would be familiar with. Bitcoin has very limited ability to perform some very basic contracts. Why is this? What is it about Bitcoin that has that property?

[0:09:10.5] EGS: Bitcoin just by its philosophy aims not to be everything to everyone. It aims to just do one thing, which is value transfer. In fact, the developers in charge, the main developer group at the moment is trying to take it away from a payments network and to make it store a value network. In some sense, Bitcoin has had its wings clipped.

It has a scripting agent that used to allow people to do all sorts of exciting things and writes different kinds of code at each transaction. But it doesn't have those features enabled at the moment. What Bitcoin is good for is money transfers and value transfers.

Ethereum has a scripting language that was designed from the get-go to operate, to facilitate smart contracts. The scripting language and Ethereum is too incomplete. You can express many more complicated things with it. It's also true that you could do similar things with Bitcoin using a whole lot of complicated contracts, but they become very cumbersome.

There are efforts to try to port smart contracts onto Bitcoin, so it's unfair to say Bitcoin doesn't support smart contracts. It could. But in the same sense that my car could be made to float if I were worked really hard at it. It's just not designed for it. From the get-go, it's designed for a different function.

[0:10:33.2] Host: I think we can all see the enormous promise and the abilities that smart contracts would give us, if we don't otherwise have. That said, smart contracts bring a whole host of security problems that are fundamentally different than most other security models. Describe for us what makes smart contracts security so different from other kinds of traditional internet security.

[0:10:54.7] EGS: Yeah, that's a great question. We all know all of the illustrious ways in which smart contract went south. There are so many that I think I was looking through a list just two nights ago. I can't even remember all the different things and the amount of money that has been lost easily in the hundreds of millions of range.

Let's see. What makes it so difficult? A couple of things; one, it's a new domain. So with a different language that people are using at the moment. That language is untested, untried, it's a little corky. So the programming language that people use to use smart contracts is one issue. The second issue is that the way these contracts are written has typically been that you issue the contract once and you don't get to change it again.

People need that certainty from you. What that also means that any bugs you have are typically encased in stone. It's very difficult to undo any mistakes you make. Unless you have thought about mechanisms ahead of time to compensate for unintended things that are happening in real-time.

It's like a moon shot. You build your smart contract up. You start trying to send it to the moon, and then it's hands off. You don't get to – unlike regular software, you don't get to go up there and wrench it around when it's not to function as expected. If Facebook has a glitch, you can be assured that there is a team of 25 people behind it trying to fix it in real-time, and they just deploy the patch without even most people realizing that there was even a glitch. You can't do that with a smart contract. Once launched, it is what it is. The only acceptable things you can do to it are those things that you would forethought place into the contract in the first place.

[0:12:40.8] Host: Got you. There had been some new kinds of security best practices that have been advocated for smart contracts, things like upgradable contracts, short-circuiting, throttling. Can you describe what these things are and what role they play in smart contract security?

[0:12:55.0] EGS: Let's see. So quite a few things have been suggested. A lot of these stem from essentially how do you detect when things are going in a direction that you don't think ought to be going, and what do you do in response? That's a large space.

I think the best example for this was the DAO hack, which really gave birth to all of these responses. I'm sure your audience knows the DAO hack all too well, so I'm not going to go into the details. I do remember the morning of the hack, and I remember waking up and my Skype was going insane, the phone was buzzing like crazy.

There was a hacker issuing these transactions that were draining the DAO and there was nothing we could do about it. You could tell that something weird was happening though. The balance was dropping and yet, you know either the thing wasn't really ready to make any – to handle any proposals at the time when it shouldn't be dropping. The balance should've been staying the same and somehow it's just going down, and everyone felt powerless.

What do you do? There are a couple of things that people have suggested. Off the top of my head, people have suggested things like what we call multi-timed contracts. That is you don't just have one contract, you crate multiple contracts and you run them in parallel. Money only comes out when three different versions all say the same thing.

This is a good way to inoculate yourself against bugs in the implementation, and the whole peer is if you have three independent implementations, they would not all fail in the same way. If you can uphold that, then you will detect when one of them is being exploited, and when that exploit is happening you can say, "Hey, there is something funny going on with this particular implementation, but the luckily the other two are keeping it in check."

This is not very different from the way we do high reliability systems, in spacecraft talking about going to moon, people will remember that the space shuttle had these five computers, right? They work as a series of vaulting computers. Every single action was replicated in all of them and they would only do something if all of them agreed – well, if a majority agreed rather. So they would be able to tolerate some number of faults.

It's the same idea except with code. You write multiple implementations and you try to essentially guard them against each other. It's a good idea. Except that, of course it might triple your costs of development and also you might not get the independence that you seek. If it's the same group of people writing the code, then the code will likely fail in the same fashion across all three of them. So that's one problem.

Let's see, another idea that has been floated is escape hatches. Escape hatches are a good idea for essentially – the core comes up in an escape hatch is this, when you detect that something is happening that should not be happening, then you bring your contract down into a more reptilian mode where it can't do anything fancy and it just shuts down in essence.

Gets into some situation where only certain activities say the owner kicking it back, for example, to function again. It can bring it back. That's a good way to ensure that, for example, if we had the DAO again, it would most certainly have an escape hatch that says, "If two-thirds of the people think something bad is going on, it should stop." This was one of the things we suggested in the aftermath of the DAO.

That's a crowd source escape hatch, where if the crowd says something bad is happening, the contract stops functioning. We were the first ones to propagate that idea. I think it's a good one. Let's see. But there are other escape hatches possible. The trigger function in the escape hatch is going to be crucial. You don't want a trigger that can be triggered by anyone and everyone, because then the certainty of operation of the contract goes away.

You can have other escape hatches too. You can have throttling. For throttling, it's slightly different. You can have triggers of the kind, if the contract balance goes down by this much over that amount of time. Or if the correspondents, if the desire to correspondents is between token is outstanding and balance is lost. These are the kinds of invariance that typically will trigger an escape hatch. Then jump into a more restricted operation where people can't withdraw anymore, and hopefully the owner can figure out what's going wrong and kick it back into the action.

Then of course, there is throttling and throttling is the idea of the payouts being delayed and limited. So that someone can't come in and empty out everything you've got and walk away with it. If that is somehow managed overtime, then you have an opportunity to monitor what's going on and perhaps intercept or freeze the throttle funds.

[SPONSOR MESSAGE]

[0:18:05.2] JM: Every second your cloud servers are running, they are costing you money. Stop paying for idle cloud instances and VMs. Control the cost of your cloud with ParkMyCloud. ParkMyCloud automatically turns off cloud resources when you don't need them. Whether you're on AWS, Azure or Google Cloud, it's easy to start saving money with ParkMyCloud.

You sign up for ParkMyCloud, you connect to your cloud provider and ParkMyCloud gives you a dashboard of all your resources, including their costs. From the dashboard, you can automatically schedule when your different cloud instances get turned on or off saving you 65% or more.

Additionally, you can manage databases, auto-scaling groups and you could setup logical groups of servers to turn off during nights and weekends when you don't need them, and you could see how much money you're saving.

Go to parkmycloud.com/sedaily to get $100 in free credit for ParkMyCloud for SE Daily listeners. ParkMyCloud is used by corporations like McDonald's, Capital One and Fox, and it saves customers' tens of thousands of dollars every month.

Go to parkmycloud.com/sedaily and cut the cost of your cloud today. That's parkmycloud.com/se daily.

[INTERVIEW CONTINUED]

[0:19:38.6] Host: Now big area of traditional security is protecting private or sensitive data. Normally, if you're not in a blockchain, you would just encrypt this data, store some security database with specialized access patterns and now you described earlier the idea of a blockchain insurance company. Let's say that you were to have such a company on the blockchain, most likely you would have some sensitive or private data that you wouldn't want everyone to have. Of course, given that in a blockchain everybody has access to the blockchain or the database. How on a blockchain would you mitigate information or privacy leakage?

[0:20:12.5] EGS: Okay. That's a very big question and a very difficult one. If we're going to be honest about the limitations of this technology, this is one of its Achilles' heels. That is privacy is

very hard to get on a blockchain. There are various different techniques for hiding certain aspects of the data you might want to hide.

Let's try to do this somewhat systematically. Let's see. If we're going to store something on the blockchain, the easiest way to make it not visible to other people is to encrypt it; that's trivial. But now, what you've done is you're using the blockchain as just data storage, highly replicated data storage. The problem that you had with privacy suddenly turned into a problem of key management. Now who gets to have that decryption key? Under what circumstances? How do you pass it from one person to another? Those all become difficult.

Since they now became – the key became an exogenous thing. It's outside the system. It's outside the blockchain. Its management is harder to control and rules about its management are not necessarily encoded on the blockchain, unless you do a - you go to great lengths and involve some smarts.

There is that issue. The other thing one can do is store private data off chain in some other system, accompanying system. One of the core ideas here came from say Filecoin and other peer-to-peer storage systems like this.

This takes the storage pressure off the blockchain. It still leaves behind the entire key management problem. There is of course the metadatation. That is, so you and I can perhaps hide what we're doing and perhaps you and I have shared some secret keys and everything we leave on the blockchain is an encrypted blob form, so if anybody looks at it they can't make sense of what's going on. But they're still metadata.

The fact that we're interacting is on the blockchain itself. If we want to hide that then what techniques do we use? That's an open research question. There are some dumb techniques, like we put up junk when we don't have anything to say to each other. But there are some smarter techniques for hiding metadata.

There's big continuum of different kinds of techniques one can use. One of the most exciting things of course is zero knowledge proofs, where someone proves something to another person without revealing any information.

Anyhow, so there are some exciting weapons in our arsenal, crypto-weapons or crypto-techniques in our arsenal for dealing with this. But there is no easy solution. When it comes to private data and blockchains, I think the two are very hard to merge. What I suspect will happen is that there will be new architectures that will emerge.

A layer in private data will have much stronger guarantees than what we have with both public and permission blockchains right now. At the moment, we just have two kinds of blockchains. As everybody knows, we have the public kind and Bitcoins, the Ethereums, etc. We have the permission blockchains where you, me and everybody in a particular industry come together and we designate some number of people as our record holders and we trust them and we trust those people to keep our data private. Maybe that's okay, but architecturally I think these two options are not the only two options and there are other ones that are about to emerge.

[0:23:43.1] Host: Can you go into more detail?

[0:23:44.8] EGS: Sure. I can tell you a little bit about some of the work we've been doing at Cornell. As I mentioned, I see two big families, the public blockchains which are wonderful. They do a lot of things for us, but everything is out in the open. The permission blockchains where once again, we have a number of people and a designated set of record holders.

I don't know that that structure, that architecture makes sense in a lot of industries. It's definitely true that you don't want to have a single record holder. We don't want to just say, "Google should hold the world's records on healthcare." That's a terrible thing to do for a whole lot of reasons.

That's bad. But the immediate reaction, the dumb thing to say to this is, "Okay, well just smear it across in different people." It's going to require a threshold T of them to make any state transition. I take my data and I store a copy at I don't know, Deloitte Touche, some other accounting firm, yet another accounting firm, etc., and five of them or 50 of them, whatever the number might be.

Then when it comes time to interact with this ensemble, I should interact with at least the majority, maybe a two-thirds majority of them, so that I can reconstitute the data. That's the standard technique. At least we have known how to do in distributed systems since the late 80s, so that 30-year-old technology. We understand as well.

From a business perspective, what have you done? Well you still have a monopoly class. You created, you took away, you were trying to avoid the monopoly while you created an oligopoly. That ensemble, you still have to pay. They're not going to do this record keeping out of the kindness of their hearts. It's a lot of compliance work, a lot of legal liability. You'll have to pay all of them.

Now you're exposing to them your metadata as well. They know a lot about you, your frequency of interaction, who you interact with and so forth. In a post-Snowden world, it's actually quite dangerous. These guys are going to be infiltrated the most and first. If one of them is infiltrated, the data gets out and suddenly your crucial business data is in the hands of people that aren't involved.

In fact, you and I when we are going into business, I think it's anathema to go and find another group of people and then trust them with our business relationship. I think that would not be my first reaction. If I'm going into business with you, I assume you're in good faith, you assume I'm in good faith, we just need some technology to help us keep the faith, if you will.

We don't necessarily need to go out and find some other people and go through those mediators for all of our interactions. At Cornell, we're working on some new technology for exactly this. The question here is can you have a blockchain of one? Can you have a blockchain where only node comes up and says, "Hey, I would like to hold your data for you, and I will do so with all the same guarantees, all the same immutability, integrity, audit ability guarantees as what you're accustomed to from both public blockchains and from mission blockchains." That is without recourse to a third party.

[0:27:05.0] Host: Yeah, that sounds like a very fascinating work. How would you deal with the – You might be able to use cryptography to ascertain that this person is actually – or this node is

actually storing your data. How would you prevent for that data from getting destroyed or going offline or just disappearing?

[0:27:18.9] EGS: You would make a lot of copies. What you also have to do in the process is as you make these copies, you have to make sure that the associated security policies with that data are always enforced. Yeah, so that's an interesting direction. I'm trying not to give away the techniques too much.

[0:27:39.6] Host: Okay, sure. We'll veer away from that so as not revealing your hand. One thing that's true, I think both in public and private blockchains is one limitation of smart contracts generally is that it's impossible to pull any data that's not on a blockchain, unless someone puts it on a blockchain.

Blockchains can make API calls. Deciding on who reports on outside data and whether to trust what they're reporting is true is a hard problem. What are the kind of solutions people have come up with for reporting off-chain data?

[0:28:11.0] EGS: This is the oracle problem. If I can step back a minute, so let's try to identify what's going on, so the listeners can put things in context. What EVM gives us, what the smart contract platforms give us at the moment is comparable to processors. It's like someone invented a replicated trustworthy processor.

That's wonderful. It's as significant as somebody inventing the microprocessor in the 70s or something of that kind. But what has not yet been invented is the runtime for the programs that execute on that processor. When I execute programs on top of a blockchain, I will want to necessarily interact with the rest of the world.

This typically means doing IO. I will want to import facts about the world. On occasion, I might want to make calls out to other services outside. There are efforts underway to make all of these things happen. The technique for importing facts into the smart contract platform is known a oracles. The question is can we build trustworthy oracles that can take facts from the outside, the external world in some way or another, and import them into a blockchain?

There has been quite a lot of work in this area. AT IC3, my colleague Ari Juels has built a system called Town Crier. The goal of Town Crier is to use secure hardware to act as a trusted intermediary that goes out and fetches data from trusted sources and gives you a certificate that says, “Well, according to Yahoo Finance, the price of GM is such and such.”

Let’s see. That’s one way of importing facts into the system. There is another technique called shelling systems, where a master student working with me built an early prototype of this. We were unable to actually deploy it. We got shutdown by Cornell’s lawyers. But essentially, what you do is you post a question to the crowd and you find out from the crowd what they think.

For example, what was the weather like in Ithaca, New York on this fine day in October? They will all say, “Oh, it’s cloudy and warm.” Then you give them a slight, small reward. As long as they know that everybody else is going to be honest, it makes sense for them to all be honest.

There are some interesting theoretic games to be played in with shelling systems. But they are a wonderful and very exciting technique for importing facts from the crowd, for holding the crowd, if you will.

The final technique of course, is to try to keep the data that you need in the system in the first place. As this entire industry matures, we’re going to find that much of what we need is already there. For example, many, many contracts need an exchange rate for some token. They would like to take action based on the exchange rate.

If we have decentralized exchanges, well then you can just consult a sister contract that’s also sitting on the same blockchain as you and get a reliable answer from that source. We’re slowly seeing the deployment of decentralized exchanges, and so we’re taking steps towards this. There are many other techniques as I mentioned for important facts from the outside world.

[0:31:32.6] Host: Right. One other difficulty that a lot of people have when writing smart contracts is that everything that happens in a smart contract must be deterministic, which is somewhat different than a normal computer. We talked about IO as API calls, but another thing is just getting a good source of randomness is actually quite tricky on a blockchain. It’s not obvious, exactly how you do that.

Can you go into a little bit detail, which might not be obvious for listeners? Why is it that every operation that occurs on a blockchain must be deterministic and what people have done to try to get randomness onto blockchains?

[0:32:05.9] EGS: Let me try to explain why it's essential. When you have all these replicated computers, it's essential that all of them agree on exactly what's happened. If your machine decides that the state of a contract is Professor Sirer did E2-E4, like he advanced his pawn. Meanwhile, I somehow trick another machine into thinking that I moved with my bishop, then I could possibly explore every single possibility of moves and then try to pick the winning game at the end of the day.

We cannot allow the state machines to diverge. If they diverge, we have what is known as a fork in the system and it has terrible consequences in that the financial history of the world disagrees. Now half your nodes believe that there was a move this way, or somebody owns $1 and half the other nodes believe that somebody also owns the same dollar. That is something you can't have.

The difficulty comes from trying to avoid this. The entire state machine has to be deterministic. You can't for example say at this junction in time, at this instruction, "Pick a random number." If you did that, then your machine would pick le's say 17, and that's your number. My machine would pick 35, and suddenly different things would happen.

For example, a lottery would pay out to a person with the lucky number 17 in your case and 35 in my case. Those are two separate people. If the guy number 17 tries to pay someone, does he have the coins? Well not according to my node he doesn't. You can't just ask for a random number by saying come up with a random number. You have to in some globally coordinated fashion come up with that random number.

Protocols for doing so securely are incredibly hard to get right. How do we come up with a random number? Well, we don't even know which nodes are part of the theory network at any one time. It's a dynamic set. It changes all the time as people drop off the network, as they

come into the network. You can't really ask everybody, so you have to somehow get that random number from some source.

The simplest thing that people have done in this space is they consult an oracle, a random number oracle. You go out off of the blockchain and someone says, "Hey, the random number is 37." There are other tricks that people have used in certain context. So if you and I want to cooperatively discover a random number, what they do is you first pick a random number in your mind. You don't tell me. I pick one too, and then you take the hash of that random number, you publish that.

I publish the hash of my random number and then we reveal. This is a commit and reveal scheme. Now we have revealed the two random numbers we had in our minds, and once I have committed the hash, I can't reveal anything other than the random number I had in my mind, because then it won't match its hash.

This is an okay scheme, except it suffers from the lightness problem. If you and I were wagering a bet. Let's say you and I will – we bet $10,000 each and you would bet even, I bet odd, and then you commit to a number, I commit to a number, we're going to X all those numbers. Well, if I see, so suppose you commit and reveal your number, and I look at this and I'm not seeing your number, I now know the outcome.

If the outcome is not in my favor, then I could actually be a jerk and not reveal my number. There I sit. In this particular example, you could maybe arrange the bet so that if I don't reveal it – this is a two-party example – if I don't reveal it, you punish me. That's okay.

There are many other instances where it's actually quite difficult to compensate for this immense power of the last revelation. If you are doing a state lottery for example and some number of people are supposed to reveal their numbers. Well, the last person to reveal his number, it knows a lot more than the others.

These commit-reveal schemes are okay in some context, but there are many others where they just fail miserably. Then you just go off-chain as I mentioned before. There are schemes for correctly picking random numbers, except they're complicated and the last, or the first correct

protocol for doing so came out about a year ago. It took us a long, long time to get that protocol right. It's quite metric.

[0:36:44.3] Host: Is it very complex procedure?

[0:36:46.0] EGS: Correct. This place is also non-trivial. Yeah, that's why it took so much on.

[SPONSOR MESSAGE]

[0:37:00.1] JM: When your application is failing on a user's device, how do you find out about that failure? Raygun lets you see every problem in your software and how to fix it. Raygun brings together crash reporting, real user monitoring, user tracking and deployment tracking.

See every error and crash affecting your users right now. Monitor your deployments to make sure that a release is not impacting users in new unexpected ways. Track your users through your application to identify the bad experiences that they are having.

Go to softwareengineeringdaily.com/raygun and get a free 14-day trial to try out Raygun and find the errors that are occurring in your applications today. Raygun is used by Microsoft, Slack and Unity to monitor their customer-facing software.

Go to softwareengineeringdailly.com/raygun and try it out for yourself.

[INTERVIEW CONTINUED]

[0:38:07.6] Host: We've gone into detail about a number of the security issues that one would face writing a correct smart contract. I've spoken with many software engineers about this and a lot of people are very pessimistic that smart contracts can never be written securely given the difficulty of writing even correct normal code, but writing correct code that natively transacts with money, and because it's on a blockchain usually by its nature, transactions can't be reversed.

It's also very hard to do insurance or risk mitigation on blockchains. Many people just believe that this is a fundamentally doomed project. What are your thoughts to this? Do you agree with this? Do you disagree with this?

[0:38:43.1] EGS: I completely disagree with that final assessment. I agree that writing non-buggy software is very difficult. It has always been difficult. I mean, I think just experimentally or just looking at empirically, but what's happened. For every thousand lines of code, you see one fatal error, and maybe many more.

I've been lucky enough to call out quite a few of these ahead of time. For the DAO, I think I was one of the co-authors on the call for a moratorium and you identified in nine different flaws with it, one of which the hacker used for bancor, we called out the front running problem, as well as the rounding issues that they have.

We've been pursuant, and indeed it's very, very difficult to write non-buggy code. In the same sense, it's very difficult to send a man to the moon and it's very difficult to make a heavier than object stay in the air. It's really tough, right? We spent like thousands of years on this earth without heavier than air object up in the air.

Kites were invented I think fairly late in the game. Let alone airplanes that carry people. That's just a very recent invention. So should we give up? Hell no. That's crazy talk. What's there to do? Well, the science of finding bugs is advancing rapidly. There is much research to be done on formal methods, on verification, on languages for expressing smart contracts that necessarily yield a correct implementation.

As well as many other things. As well as bug bounties, multi-time implementations, escape hatches and so forth, practical measures to make sure that yes, there will be bugs. But we will manage to deal with them.

Many of these things that people want to do can be adequately addressed with the technology we have even today. I'm not stressed about this. Yes, there will always be bugs. We've always had buggy systems, and yet we managed to build systems that are robust in the face of bugs.

[0:40:50.1] Host: Right. Speaking then of buggy systems, the most popular language for some of our contracts on Ethereum is solidity, which is used to write the vast majority of the theorems for contracts. After the parity had, which is a 32 million dollar hack of a multi-sig wallet, solidity in the EVM fell under loads of criticism for being poorly designed and having unsecure defaults. You wrote about this in your own blog post covering the parity hack. What are your thoughts about the design general of solidity specifically as a program language and of the EVM as the underlying virtual machine?

[0:41:21.3] EGS: The EVM has been particularly resilient. I have not really said anything about the EVM. Solidity is an interesting case study. The fundamental tension there is between having a language that looks familiar, that has good aesthetics, that is easy to use, versus a language that is easy to write secure code in.

Secure code doesn't have to be at odds with easy to write, easy to understand code. But practically, that seems to be the fundamental tradeoff. Okay, those are the two things. Complicating the story is the fact that not only do you have to design the language, but you have to communicate its quirks to all of the practitioners.

There is a documentation and communication challenge with any new language to be used. A lot of the bugs that we saw with solidity were things that were well-known. The reentrancy bug was well-known, that's easy. The parity missing private modifier, that's well known. There are many other things like this where we knew that the certain anti-pattering should never be encountered, but it wasn't communicated well enough.

What do I think about solidity? I think it's actually a pretty darn good effort at finding a language that is easy to use. I like that language. I have a bunch of gripes about it, but at its core the tradeoffs they made are quite sensible. I like the aesthetics, I like the way it looks.

It managed to get into the game a whole lot of programmers who would've put off if you suddenly said, "Well, before you write anything at all, you have to first take a course in logic followed by a six-month crash course, follow-on course on program verification." I think that would've been the non-starter.

Moving fast and breaking things is very much part of the credo here. It's going to happen. You just have to have sort of a healthy attitude about these things where you say, "Look, this is normal." On occasion there might be missed steps. In the early days, you can path up for them in using some measures. At some point, it's going to be less and less easy. In fact, it's going way more difficult to compensate.

Solidity in my view is as I said, a decent tradeoff. There is much to be improved in solidity. I would like to see much better types. The type system is a little weird. People have to use the safe math of floating points, or rather fixed point is not supported well. There are a bunch of other issues. The defaults are a little long key. They should really default to internal and not visible to everybody else. These are things that one can tweak within future revisions.

I would not put the blame on the developers. This is not an easy thing to build. It's many, many dozens of people who work on it. Yeah, and there will be missed steps. It's just part of the game, and the early adopters will see it and they will have to cope with it. That's just normal.

[0:44:27.0] Host: Right. This might be an entry on follow-on question, but how –if it were up to you, how would an ideal smart contract program and language be designed?

[0:44:36.6] EGS: That's an interesting question. One I've thought about a little bit. This is far from my field. I am a systems person. I do distributed systems. I'm not a programming language person. I can give you the high-level properties. I would love to be able to understand in variance associated with the code upfront.

I can't do that if the language is – if the contract is just specified mechanistically. Maybe here is an anecdote, I think instead of being too abstract. Let me tell you about what I see when I teach systems. I teach an operating systems course. It's typically the first time any of our students build something complicated. They build concurrent systems that are supposed to be correct under all conditions.

They give you a buggy implementation. You flag it. We flag it. Out of 25 points, if it's buggy you get zero or 1. Okay, no more than that. If it's buggy, it's really harsh. Then of course you have the students immediately in your office, and you say, "Look, you did this, this, this wrong. Clearly

– or not clearly. I'm worried that the situation is something that shouldn't happen. Two cars are colliding on top of your one-lane bridge. This should not be happening."

Then the student tells you, "Well, that can't happen, because for that to happen this light has to go green and that's guarded by this other thing, which is guarded by this thing." You get this incredibly lone reasoning for why that bad situation cannot arise.

Now keep your eyes peeled for this, just about every single smart contract security proof is essentially a long litany of these kinds of, I don't know what. It is like the screeds or whatever the word is. Somebody just whining on and on about how the bad situation cannot arise.

Altogether, it's just not convincing. I don't want to go through that logical process. What they're doing in that case to be technical for your audience is they're giving you a temporal logic proof, let they're saying that particular event or scenario cannot happen, because it's – essentially like, for it to happen, and this has to happen beforehand da, da, da, and the precondition will never happen.

I don't have enough life expectancy to potentially enumerate all of those scenarios. They're exponentially many in number, and rule them all out. The language has to rule them out for me. I want to be able to say, "I'm worried about the following events. This is my invariant and I wanted to protect it at all times." For example, for the DAO, the number of tokens and the amount of money that it has to have a correspondence. It better ensure that for me. If it's not doing that, there is something enormously broken.

A good language would allow me to express that. Solidity at the moment has something like and it has these requires and so forth. But it's not exactly what I have in mind. That's what I would really love to have.

The other thing that's cool, that I know about, that most people haven't thought about is backwards reasoning. I would love to see this in a language, so any of your listeners are thinking about it. In the 90s, or in the late 80s or early 90s when software safety was really big and people were building embedded systems, they tried – they came up with languages where you can execute backwards.

I would say something like, okay I know one specific case. The BMW anti-lock braking system. There are some scenarios that should never happen. Can we start in that bad scenario and execute backwards and find out an initial scenario that can trigger it? Isn't that kind of cool? I think it's neat.

People also apply this to TCAS, which is the system that makes sure that the planes don't hit each other in the sky. I don't know if you ever heard it. I hope nobody ever did, but I'm sure someone will have heard this. Sometimes, it actually blares out in a loud voice, it tells the pilots what to do. It says things like, "Pull up. Pull up." In fact, it says it with greater urgency. So, "Pull up," like it really shouts.

If it's telling you to pull up, it should be telling the other guy to push down. In some cases, it might change its mind, because if the other guy is doing something counter to what he's being told. In the TCAS system, you don't want a collision course. There were extensive studies of backwards execution, trying to figure out for a given implementation will it ever and what are the conditions under which it will tell a bad thing to a pilot.

[0:49:19.1] Host: Interesting. That backwards execution, it doesn't cause this combanitorial explosion?

[0:49:23.5] EGS: Sure. That's why it's hard.

[0:49:25.5] Host: I see. Okay, got you. Got you.

[0:49:27.9] EGS: You need to couple it with a lot of techniques for trimming down the states space. Of course, I mean on this topic there are many other things we can do. There is symbolic execution, like there is an entire area of model checking that people have worked on, where they can algebraically without having to explore every option, they can capture what would happen and then make faster progress through the state space of possibilities.

Anyway, it's just a fascinating area. Tools that's helped people, that's helped program designers, smart contract designers in this regard would be wonderful tools that are coupled well into the language, or built into the language would be fantastic.

I expected to see these for solidity. I don't see solidity syntax or feature set as a debilitating thing. We don't have to scrap it and come up with something new, but we just have to evolve it towards a better foundation.

[0:50:22.3] Host: Right. That makes sense. I want to shift gears a little bit and – so we talked a about the abstract security properties of smart contracts. But I want to talk a little bit about what kind of security we see in practice out in the wild. You spoke earlier about to the biggest hacks in Ethereum history, which were the DAO hack and the parity hack.

Famously, you presaged the DAO hack and you wrote about the parity one. How would you say the community has evolved in light of these attacks, and how has the average security hygiene or sophistication changed since then?

[0:50:56.2] EGS: The DAO hack was monumental. It's really changed the way people approach the entire smart contract ICO space. That people understood that it was difficult to write these smart contracts. You couldn't just sit down, go on a bender and write some java script code and put it on the blockchain and have it retain some value.

What has changed since then? I think there is a much better understanding of the quirks of the solidity language and some of the VM runtime quirks. There's been a lot of efforts spent on practical measures. As I mentioned, bug bounties, multi-timed implementations and escape hatches. Just about every smart contract I looked at has had some kind of an escaped hatch built into it. That's been wonderful to see. I think it's irresponsible to build things without escape hatches.

That's what has changed. Has anything else changed at a higher level? I don't know. The design process, I don't know if that has changed. I think our people writing qualitatively better smart contracts? Yes, they're better than they were at the DAO, at the time of the DAO. Are we

secure? Are we there yet? No. By no means are we anywhere close? Are we going to see more exploits? Yeah, for sure. There will be many. It will be a lot of fun.

[0:52:25.3] Host: Right. Fair enough.

[0:52:25.9] EGS: It will be and it will be interesting. Yeah.

[0:52:29.3] Host: Are there anything in particular that you'd like to see be more adopted as security best practices?

[0:52:33.9] EGS: Bug bounties would be wonderful to have. We should see more of them. There are lots and lots of smart people out there and we're not tapping into their collective smarts as well as we should be. Let's see. What other best practices? There are tons of things that people are doing kind of weird. Not wrong, but weird.

Again, I looked at the bancor code recently and they have their own math routines. That kind of stuff should be in the system. Nobody should be using safe math, nobody should be using their own approximation of a log function or whatever else. These things should be provided. Somebody should do them correctly and once and for all, and we should never ever have to audit that kind of code again. Bancor had all these magic numbers smeared all over the place, and it turned out that some of them was not incorrect.

Yeah, this is best practice-wise, the more is in the system, the better it is. The more somebody spends effort to get it right once and for all and build it into the runtime into the language, the better off we all are.

[0:53:40.3] Host: Right. Makes sense. Zooming out a little bit, one fundamental limiting factor for the wide adoption and usage of cryptocurrencies is the difficulty of key management. A lot of people believe that most normal people will just never be able to do this well enough to be able use cryptocurrencies realistically. How do you see key management as a security problem evolving?

[0:54:04.9] EGS: That pessimistic stance is what we started out with when we were first looking at Bitcoin. I'm sure everybody knows this, but Bitcoin thefts are happening all the time. People lose their keys constantly. There is no shortage of messages on bulletin boards saying, "Hey, I lost so much money. I lost so much other money."

Then there is the litany of responses, which is, "Sorry for your loss." Then it gets repeated so many times that it has gotten contracted down to SFYL. An SFYL event is like a thing that we talk about. What happened, they had an SFYL event, a sorry for your loss event.

Where is this stemming from? It's stemming from the fact that for many years, the developers essentially outsourced the security task to other developers. Instead of securing the cryptocurrency, using the cryptocurrency system, it didn't occur to them that they could take on that task. They always thought, "Well, you know if your phone is hacked, it's your fault. Or if your laptop is hacked, it's your fault."

Might be. I mean, it is a fault for sure. But if you begin to say that, if that's your mentality then what you're really saying is these systems will always be beholden to the security of the underlying competitional platforms; our phones, our laptops. Those things are nowhere near secure enough for high-value assets, like crypto.

Your regular old Verizon branded Android phone is way out of date in terms of security patches. If you're using one of those and you're holding coins, you will get hacked. You're going to install a stupid flashlight application. It's going to ask for every permission on earth and you want that light, so you'll say yes. Then the next thing you know, it's sending your coins to whoever or someone.

This is just the way of the – or it was the way of crypto and it seemed really hopeless. But Itayayel and myself, Itayayel is now a professor at The Technion. He was a post doc at Cornell. Itayayel and myself, we took a look at this whole situation. We said, "Look, it doesn't need to be this bad. We can protect some of these coins and we can invent mechanisms that are actually helpful in stemming attacks of this kind."

The simplest thing that we came up with was called covenants and vaults. A vault is a mechanism for protecting your coins that are not in motion. When you are not going to use your coins, you put them in a vault. If the hacker breaks in and moves your coins, then you get to undo the hack. To pay someone, you have to unvault your coins. You can't just pay out of a vault. The finality of Bitcoin transactions is not affected.

Yet, if somebody steals your coins, you get them back from the hacker. It is a pretty cool feature. Anyhow, so there are actual advances that's helped the state of the world. Of course, the Bitcoin world is so busy with the ongoing craziness with the block size debate that they haven't had time to do anything except very convoluted complicated hack called Segwit.

They haven't adopted this yet, but we've seen adoption in other areas in other systems. But there are things we can do. I would urge the developer community to start looking into actual encrypto mechanisms for better security. Do not rely on the Android developers to give you a perfect operating system. That is never going to happen.

Do not expect a laptop to be uncompromisable. Never going to happen. It's 50 million lines of code in there, of course there will be flaws and bugs and so forth. Plus more that users actually install. Don't wait for those platforms. It's not going to happen that way. You're going to have to build in features to protect your users.

[0:57:59.5] Host: Got you. Let's again shift gears one more time and look toward the future of blockchains. This is something that you write quite a lot about. Interest in blockchains and cryptocurrencies have it feeder pitch, and it's to the point now where a lot of massive companies and regulators are starting to wet their appetite and want to get in for a piece. Are there any key events that you're looking for on the horizon that you think are going to be major inflexion points in the future of blockchains?

[0:58:27.5] EGS: Sure. Regulation of the exchanges. At the moment, there are a large number of exchanges that's running wild. They operate out of funny jurisdiction, nobody knows exactly where they are, what laws they're a subject to, nobody knows their internal state. There are lots of indications that they might be doing something funny.

When we see the regulators move in and start trimming that kind of behavior, start intervening and exchanges gone wild, then we will see a different – a shift. The money that is currently on the sidelines, it comes in only to the extent that the holders are greedy. They look at what's happening. They're like, "Hey, I want a piece of this action." They'll take the enormous risk of being bamboozled at an exchange.

There is some money of that kind, the greed, the risk-happy greedy people. But there is a lot more money on the sidelines, or there is a magnitude more in the hands of people who are looking at this saying, "This is a legitimate area." All of the geeky crowd is saying this is really exciting. There has to be something here. Except the risks are too much, the markets are too unregulated. In fact, the SEC pronouncement was had everything to do with just how terrible the exchanges were and how –

[0:59:50.7] Host: Sorry, can you go to detail. What was that SEC real briefly?

[0:59:54.1] EGS: When the SEC ended up clamping down on the Winklevoss ETF, proposed ETF, they said that an ETF at this point in time would be a bad idea, because the exchanges are unregulated and there are all sorts of shenanigans that people pull at the exchanges, such that exposing your regular retail investors to the kinds of exchanges that currently run the space, would be a falling. That was their reasoning.

You can argue all you like about SEC. Nobody likes regulators, right? Nobody wants them to come in etc. You can say, "Well, the system is going to self-regulate and so forth." I definitely am not a fan of hard regulation. But if you ask me what is the next milestone on the regulation front, them stepping in to curve the excess, the crazy shenanigans at the exchanges is going to be the next big, big step. It will drive away some of the money, some of the illicit might go elsewhere and I'm perfectly okay with that. It will bring in – if done right, it will bring in a lot more money that's sitting on the sidelines.

[1:01:02.4] Host: Right. That makes sense.

[1:01:04.2] EGS: There is much more clean money in the world than there is illicit flows.

[1:01:08.5] Host: True. Very true. In your blog as you're writing a lot about cryptocurrencies. You say that your mission in your blog is to be a bulwark against the rampant misinformation and foggy thinking in the world. Now cryptocurrencies have had a massive influx of attention in the last year, and a lot of new people answering into the cryptocurrency community, most of whom are less informed and less technical. Maybe they care more about trading and speculating, they don't care about technology. Do you think this has –

Let me ask a two-pronged question. The first is that how do you think this changes the culture and the progress of cryptocurrencies? Also, do you think it changes the prospects for its success?

[1:01:47.1] EGS: Yes, both in a very positive direction. The new people coming in, they don't have the baggage of the old-timers. The come in with an open mind. They typically come in, because they see something exciting and/or they see the opportunity for making money. Both are fine. That crowd typically comes in with a more open mind than somebody who has been, I don't know, into Bitcoin since 2011.

What I see is a big problem in these systems. At their heart, they are open source projects with a user community. They're valuable to the extent of their user community. The sum total of wealth inside Bitcoin is capped by the amount of money people have put into it. It's capped by the number of people who are excited about Bitcoin.

New people coming in with a fresh viewpoint is always good. New people coming in building new services on top is a wonderful thing to have. That will displace the toxic situations that we sometimes see in communities. Maximalism is always a bad thing to have. Infighting is always a bad thing. No matter what side of the block size debate somebody might be on, big or small blocker, nobody is happy about block size debate. It's just nothing is just – no joy. It's just not good.

Cutting that down is always good. The best way to cut it down is to bring in a whole bunch of people who are going to say, "Guy, let's just find a block size and move on because there is so much more to do on top." That's I think exciting.

[1:03:30.7] Host: Right. Well that's really great to hear that kind of optimism from you. I'll have to take that with me. I think we've come up on the end of time, but I want to thank you so much Gun for taking the time to chat with us. It's been a fascinating conversation.

[1:03:43.8] EGS: Thank you so much for having me. It was indeed a lot of fun.

[END OF INTERVIEW]

[1:03:50.2] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at symphono.com/sedaily. That's symphono.com/sedaily.

Thanks to Symphono for being a sponsor of Software Engineering Daily for almost a year now. Your continued support allows us to deliver content to the listeners on a regular basis.

[END]