EPISODE 1179

[INTRODUCTION]

[00:00:00] JM: Osquery is a tool for providing visibility into operating system endpoints. It is a flexible tool developed originally at Facebook. Ganesh Pai is the founder of Uptycs, a company that uses osquery to find threats and malicious activity occurring across nodes. Ganesh joins the show to talk about osquery usage and his work on Uptycs.

Ganesh, welcome to the show.

[00:00:28] GP: Jeff, thank you for having me here.

[00:00:30] JM: Today we're talking about osquery. Can you explain osquery from a high level just what it does and why it's important?

[00:00:37] GP: Osquery is an open source agent which was introduced into public domain by Facebook early 2015 or thereabouts approximately. It is an agent which can be used for multiple security and IT visibility related use cases. It was built on the premise that if you use SQL as an abstraction to pull out information from an operating system endpoint, and what gets made available by the agent itself is a bunch of building blocks from the operating system. As in who are the users who are currently logged in? What's the inventory of packages? What kind of software processes are running? What kind of socket connections have been established? Each building block, whether it is the behavioral aspect or the configured aspect of an operating system, it's abstracted and presented as a series of tables, and osquery as an agent makes all of those available for people to query and then draw conclusions. So that's what it's intended purpose of. Think of it as a very high-fidelity agent to provide telemetry for a wide variety of security and management related use cases.

[00:01:53] JM: Can you say more about the range and type of data that we can get from osquery?

[00:01:58] GP: Yes. The nice thing about osquery is that it was built day one so that it runs across a wide variety of operating systems. Today as we speak, it nominally covers various flavors of macOS, Windows and multiple flavors of Linux, and it's somewhat straightforward to port to other Unix variants. As well at a high level you can think of what is abstracted and presented as a series of tabular telemetry to be falling into two buckets. The first bucket is around configured state of an operating system. For example, if you might want to know what's the major, minor, and revision, or diversion of operating system. What kind of packages are deployed on a Linux machine? You typically have a series of RPM or Debian commands that when you invoke on an operating system these are presented to you when you run these commands at a shell.

What osquery abstracts is it pulls out the static or semi-static pieces of telemetry. Converts them into really nice structured telemetry and presents it to whoever invokes it. So that's the first category of information that it presents. The second category is around the behavioral aspects of the system, which is to say that osquery integrates using the operating system provided APIs and is able to tap in to the behavioral aspects, which can be captured using system call activity. As an example very specifically on Linux, either by using what's called as the netlink key audit interface, or in modern versions of Linux using what's called as an eBPF probe, you're able to capture information which is leading indicators to say that did someone log into the system? Did someone establish outbound connection by doing a DNS lookup? Did a process get launched? If a process got launched, what was the checksum of the executable which got launched, right? Pieces of information which are effectively behavioral changes to the system because the system is alive and running, this is the second category of event-related behavior that osquery is able to capture. And if you look today in public domain you can see that they're probably around 200 to maybe 400 different tables depending on the operating system type and combinations. That's made available for end users to query and extract information from.

[00:04:38] JM: Does osquery lower the barrier to entry from making queries for making interfaces to the infrastructure of an overall system?

[00:04:48] GP: Yeah. It most certainly does. One of the nice aspects about its interface is that the medium to ask the questions and get the response is all made available using a SQL interface. What that really means is if you look at the implementation, it uses a variation of – Not a variation. It's an ANSI standard SQL, but based on top of SQLite database, but there's no real database per se. The medium to define table, the medium to ask questions to say that select star from processes. What it really means is that find out all details about all processes which are running. What's the parent ID? The parent process ID. The resident memory size. What's the current CPU usage? All these pieces of information which you might otherwise get when you run a command, they are abstracted and presented nicely using a schema.

The benefits of the schema is that anyone who is somewhat comfortable with SQL has the ability to ask questions and learn about what a process or a socket or operating system runtime means without being an actual operating system expert. And once they learn the dialect, they have the ability to ask the same questions of Windows, Linux or Mac without having to worry what exactly is the actual representation and behavior on a real operating system instance, right? So this abstraction kind of lowers, if you will, the barrier when it comes to pulling telemetry out, because you're now talking in a language of data extraction versus using the actual language of the operating system to pull the same information out.

[00:06:38] JM: Can you say a bit more about the abstractions and the data structures that osquery presents to the user?

[00:06:46] GP: Yeah. Just to put things into a little bit of a perspective. If osquery didn't exist, there were many approaches that developers and operations engineers and security engineers used to pull information out. So as an example on Windows, you might use a tooling such as Sysmon, which captures the behavioral aspects of a system and writes it out into a series of lines, which are effectively log file. On Linux machines, those who are experts, they might know how to invoke various commands. And if you're a little bit of an advanced user, you might walk what's called as the Procfile system to figure out how exactly is the operating system configured and where are things running right now.

Now, osquery has been built in a way such that the frontend to ask questions is SQL. And of course the SQL can be invoked either over a command line or through HTTP or a TLS-based connection. But in all cases, when the request arrives at the heart of the osquery engine, it in turn goes through abstraction where there is a virtual representation of a table which maps to the information that's being requested. For example, processes, process events, which is event-driven version of processes, or sockets, or the etc hosts file. All of these artifacts that can be presented back to an end user are represented as tables. And those tables are coded into the SQLite database and they're really virtual tables, which means that there's no real database out there.

And each time a data structure which represents this table is being invoked, there are virtual callback hooks which are plugged into the executable of osquery, which is behind the SQLite implementation as callback. And those callbacks in turn use the native operating system APIs to pull the information out and provide the data in a tabular format, which effectively is represented as a table, which essentially represents the data structure that the engineer, whether an operations engineer or a security engineer, is trying to look at, or an analyst is trying to look at to draw some conclusions.

[00:09:13] JM: Can you go into a few examples of why osquery is useful?

[00:09:18] GP: Yes. Osquery by itself is an agent, which, when you instantiate, it doesn't do much until you configure it a couple different ways. Most notably it is useful for extracting information, as in telemetry, by asking a series of scheduled questions or configuring its internals so that it can capture the behavioral changes in a system. Now given the range of what it has to offer, the use cases can go as wide as getting information about asset inventory all the way into advanced behavioral detection for intrusion detection, file integrity monitoring and a wide variety of use cases. At a high level, as applied to the domain of security given the breadth, I'll just focus in specifically on security as an example. It is extremely useful for both proactive as well as reactive security. By proactive security, I mean, the ability of an organization to ask inventory related questions about their fleet of machines and resources so

that they can proactively, by looking at the information coming back, audit and figure out misconfigurations in their entire fleet, which is to say that by asking the right series of questions one is able to discern if the disks have been encrypted. Has the firewall been enabled, right? Answering such questions and then potentially remediation based on the findings of such questions allows an organization to proactively improve their security posture and address basic security hygiene in a meaningful way, because the telemetry provided by osquery facilitates these variety of use cases, right?

So from a proactive perspective, one can broadly think of audit and compliance as two key use cases that can be addressed by pulling telemetry out from osquery and aggregating it and looking at the data to draw conclusions. On the other hand, for reactive scenarios where you're trying to like discern and find out what are the changes happening on a machine, and are these changes indicative of anything potentially malicious or if there is something which is going to be a detection based on the telemetry which is noteworthy? Osquery also facilitates that.

To facilitate those type of use cases, what osquery provides for you is behavioral telemetry of the system. For example, if you're trying to figure out something which is a deviation from a baseline across your fleet of machines to say that, “If this happens, then there is something potentially malicious. Here is how osquery could help you.” Say on a set of machines in the cloud that osquery is detecting that there was an outbound socket connection established at 3 a.m. in the morning, and what you have is information saying which process? What IP address and potentially you know the DNS lookup which is done are pieces of relevant telemetry which are published by osquery.

Now these pieces of telemetry, when you combine it with a reputation database such as IP reputation or a domain reputation database to say that, “Yes, a connection was established at 3 a.m. in the morning, but the IP address happens to be of a well-known command and control center.” Then you know that there might be something malicious going on, because what the operating system told you as an event could now be correlated with a known bad reputation database to like establish that maybe this is a noteworthy signal that someone in the SOC team or someone in the CSIRT team should be paying attention to this.

There are other use cases as well. Example, osquery has a solid approach to monitor for file system changes. Especially if you're under compliance regimes such as PCI and others, you can configure osquery to monitor, read, write and update activity or change activity related to the file system directory and individually at a file level. And if there are certain set of files, if you want to know if something got written or something got altered, and as soon as you detect a change, you essentially send that up to system as a signal to say that, “Here's a change.” And someone can draw conclusions from that change.

Essentially, osquery facilitates reactive detection type scenarios, right? So between proactive telemetry and reactive telemetry. It lays the sound foundation with high-fidelity to build your security program and hence measures your security posture if you will.

[00:14:24] JM: Can you say more about the storage model for data that gets put into osquery?

[00:14:28] GP: Yeah. Osquery is built on the premise that predominantly it behaves as a sensor or agent to pull information and send it out based on configuration which has been provided to it. A popular model is to schedule a series of queries so that you can pull static information, which doesn't vary as a function of time, and typically write it to a file or write it to one of the many plugins which are available, one of which happens to be a HTTP TLS plugin so that you can send it to a backend where someone can do something with that collected data.

So in those scenarios where you're pulling something out as a part of a scheduled query, it has a local storage based on RocksDB. And the reason that gets used is that osquery can be made to operate in an efficient manner, which means that when you ask a question about a specific table, you can ask it to operate in a snapshot mode, which is to say that, “Tell me everything that I need to know now,” and it will transmit the information in entirety. And if it is operating in a differential mode, it then has the ability to say that based on the last time I provided information to you, I'm going to be providing add and delete changes, right? So that it's a little bit efficient as opposed to transmitting all the information back. It can send the incremental changes of additions and deletions.

In order to facilitate that, it has a local database based on RocksDB, and that is what it uses as a backing store. Now that backing store is also used for the purpose that if your fleet of osquery agents when deployed across machines are not communicating with the backend due to some kind of resource or connectivity issue, there is the ability to store that data so that it can be drained out subsequently when a connection is established. But the local data store is primarily used as a temporary basis for these scenarios that I outline. The expectation that the actual telemetry so that it can be actioned and used for security of management use cases is really based on the fact that the output of these queries get written to a file or get sent over a TLS connection so that someone can do something useful with that data by loading it into some kind of an analytics platform.

[00:17:10] JM: So do I typically want to install osquery on a single machine or am I installing it across an entire fleet of my machines?

[00:17:18] GP: That is actually a terrific question. As a part of evaluation as an engineer of sheer curiosity, to be amazed, you might deploy it in a single machine and discover that the fidelity of telemetry and the breadth of telemetry and the potential use cases are immense. But to pragmatically use it, one typically deploys it on a fleet of thousands or based on our own experience across hundreds of thousands of machines. And once you have the ability to like orchestrate these fleet of osquery deployment, which typically is one osquery instance per operating system instance, or in the context of a Kubernetes type deployment, it's based on one daemon set, which typically is one per node, if you will, in a privilege mode. The osquery, once activated, is both pulling out scheduled queries and it is also pulling out information about the behavioral changes of the system. Once you have this information in aggregate, it facilitates a wide variety of use cases.

To put things into perspective, if your charter is to do simple cohort analysis, which is to say that if you have a thousand database servers and one of the servers exhibited outlier behavior, which could potentially be anomalous in nature if it at 4 a.m. in the morning a domain look up for a domain which has got a bad – A certain known reputation and then it established a socket

and then there was a data which was exfiltrated and a file which was downloaded in a location and then a process was launched, right?

Now, you've seen based on osquery telemetry that on a single machine, all of these things happen. If you have the ability to like in aggregate look across the telemetry from all of these machines, you then have an approach effectively to do cohort detection to say that, “As soon as you saw one machine initiate an outbound connection and you have all the telemetry around what that outbound connection look like,” you then have the ability using osqueries, fleet management, or if you build some platform to orchestrate a series of requests and responses with osquery, you have an environment to quickly turn around and ask the remaining 999 machines, “Did you, in this window of time, or do you right now see the same behavior that I saw on one of the database servers?” And if the remaining 999 machines were part of a database cohort group and they're supposed to be behaving identically and if the response is no, then you likely know that there is something one-off which is happening, right? So the benefit of deploying it across the fleet is that you can get the telemetry trends and try to figure out by looking at the data to say what's wrong. Or when you operate them together as a singular unit or as a single detection network, you have the ability to ask questions and facilitate almost real-time cohort analysis for doing some kind of outlier detection, if you will.

[00:20:28] JM: So you run a company called Uplytics. What does Uplytics do?

[00:20:32] GP: It's Uptycs. Close enough. Uptycs –

[00:20:35] JM: I’m sorry.

[00:20:35] GP: No worries at all. Uptycs, the name came – It’s a very engineering-centric name. For those in your audience who are familiar with uptime, it is a unit measure, a quick measure of how long a system has been up. We started off with the intent to provide a simple notion of uptime analytics, and hence the name Uptycs, if you will. But at Uptycs what we provide is a security analytics platform. And the security analytics platform provides coverage for a variety of assets and resources which are of importance primarily to modern cloud native

organizations and directionally aligned with where even large enterprises are heading. Notably to say that we provide coverage across your laptops, as in Windows and macOS to provide visibility into your production endpoints, which is where hopefully organizationally whether it's the creative work of developing software or documenting or your sales team and all might be working on.

And then the environment where for most organization where the crown jewels of the organization run, as in the production endpoints, typically ends up being a Linux or in some cases a windows server either in a data center. Or 99% of the scenarios that we encounter these days is predominantly in the cloud in either AWS or GCP, right? So and the package in this cloud environments is either a virtual machine or a container.

So at Uptycs, what we do is by leveraging osquery. And we've built inspired by osquery other queries called as cloud query, cube query and SaaS query, which are also gathering information from these other sources. But our analytics platform captures telemetry from these operating system runtimes as well as the cloud environments where our customers' productivity and production endpoints run. And then we provide a medium to help them do both audit and compliance as well as detection, investigation and response type use cases. That's what Uptycs does. One way for you to think if you're more in line with how the industry analyst thinks of us, we provide a lot of capabilities around what's called as extended detection and response, or arguably on the cloud side, the cloud workload protection if you will. That's the type of solution that we provide for our customers.

[00:23:16] JM: So what were you setting out to accomplish when you started to build up Uptycs?

[00:23:20] GP: Yeah. So at Uptycs, we were inspired in the past based on a tooling that we saw, which was used for debugging diagnostics. Very specifically based on prior engagement, there was a system called as the query system, which provided a lot of visibility. In 2016 when the venture was formed, we were excited to see if we could take this approach of doing

debugging diagnostics and instead bring it into the domain of cyber security so that we can empower a wide variety of use cases using a platform-centric approach.

So we did a couple of things, which is in line with our very tech-centric and engineering background. We set about to build a platform from day one, which allows us to capture telemetry at source using a structured manner, and the implications of which is that our pipeline is ETL-free, which is to say that we came up with an approach which is extremely cost efficient and didn't incur the traditional overhead of extract, transform and load that one might undergo if they decide to just send logs as opposed to sending structured telemetry.

So in our case, when we built the platform, the intent was that the sound basis of structured telemetry, you can build a platform, where if you envision the power of relational algebra and SQL, it would facilitate a wide variety of use cases in the domain of security. And much to our thought around this approach, and more importantly, inspired by the customer engagement, I'm very fortunate to let that that approach has been successful. It doesn't really mean that we expect our customers to use SQL day-in and day-out. But the expectation is that if the insights that we graphically present out of our platform do not suffice to majority of our customers, whether they are security engineers, SOC analysts, security architects, they get the comfort that everything is very well-structured. It is very visible. What you see is what you get. And what you don't know yet can be extracted by just doing one SQL join even if Uptycs didn't have the forethought up front to begin with and present that insight.

Point is that we have a platform which facilitates really quick time to insights, or how our customers measure it is mean time to detection or mean time to response. And if there are things which we haven't thought of yet because the telemetry is structured and it's all there, the customer or our professional services then is able to like extract these pieces of information out given how we've engineered the platform, right?

So we're very fortunate that day one what we set out four plus years ago, we've truly been able to build this data engineering pipeline with the relational algebra and SQL as a basis, and that's

paid handsome dividends when it comes to our customer engagements and giving them comfort that it's data-driven analytics.

[00:26:40] JM: Can you talk a little bit more about the features that you're trying to build with Uptycs, like detection rule set and file scanning and stuff?

[00:26:48] GP: Yes. So at Uptycs what we've built is a data engineering pipeline. There are stages in the pipeline. You can think of the stage one as being the collect stage of the pipeline. Here, our approach, which is probably not necessarily at odds, but very different from what might be the norm out there, is that we believe in a completely structured world, which is to say that telemetry, we try to abstract and convert it into a table and we try to pull it out in a structured manner as soon as we can all the way from the agent. Now this allows us to precisely pull information which is needed as opposed to taking a log file and then trying to impart structure later. If there are two fields out of the file from the source, we have the ability to pull this out, right? So pulling information out using various query-driven tools that we're building in addition to osquery lays the foundation for the first stage of our data engineering pipeline.

Now the second stage of our pipeline is the ability to aggregate many endpoints and start looking at the data so that we can do a little bit of the cohort analysis that I was talking about earlier, which is to say that each osquery endpoint or an integration that we use in the platform is 24/7 live connected to our backend, which is a multi-tenant SaaS service. And for very large enterprise customers, we can also instantiate it to on-prem especially if you have regulatory needs or if you need to run an AWS GovCloud or for wide variety of other reasons.

But the second stage, which is the aggregation stage of the pipeline that we've built, we've modeled it to resemble extended detection network. What this allows us to do is not only ingest massive amount of telemetry much like how a content delivery network operates using various hashing techniques to spread the load out and ingest the data, but it builds the sound basis for internal platforms so that we can do streaming analytics on the data, which is being sent to our platform. And that's where all of this feeds into the analytics stage, which is the

analyze stage of the platform, which is the third stage. And this is where very specifically to a question, when the data which is being streamed from osquery or from other integrations that we provide, query-driven integrations we provide, arrives at the platform, while the data is in flight, we have the ability to do high-fidelity correlations using what we call as Lambda analytics. And the idea here is that using a markup language that we've built, it allows us to take these pieces of telemetry which is coming in and transform this telemetry into what we call as a signal. And subsequently we correlate a set of related signals to say that do these signals provide sufficient value to be deemed as a detection? And a detection is something which is noteworthy then that can be presented using a standard framework such as MITRE ATTCK and present it to a SOC analyst, right?

So what the pipeline does is it facilitates collection of telemetry in a structured manner. Pass it through our aggregation to get the scale and to do the cohort analysis. And finally transform this data from billions of pieces of telemetry occurring in isolation across different fleets and resources. It transforms it into a smaller subset of signals, and the signals then get subsequently trimmed down into a series of detections. And a detection is something that we present. In a similar way, the analog of this on the proactive side is that we can ask a series of questions for audit and compliance. Data flows through the same platform. We subject it to a similar set of audit detection rules, and then we package them into results and present the findings using CIS, PCI and other compliance-related approaches, right?

But think of what we do for a second. If you were to park it and say that this is not – If I were to give you a simple example, there is a lot of precedence for what Uptycs does and it is inspired by what is probably very common in the ad tech industry. Today when you try to go through the act of buying something on a web browser and you add things into a shopping cart and you decide to close the browser tab, someone might present a quick coupon or say, “Yes, $10 off to make you close the transaction today.” The reason they are able to do that is that the behavioral events, which can be captured from a web browser, are transmitted in real-time to ad tech, ad decision engine at the backend, which is quickly reacting to the behavioral change of the browser and saying, “Here's what you should be doing next.”

In a similar way we use osquery and structured data and push it into our platform and our streaming analytics does the transformation to say that the telemetry that I received, is this a high-fidelity signal and are there related signals, which when viewed and aggregate correlate to something being a noteworthy detection? Hopefully that made sense as I described as to how we take telemetry and transform it into signals and transform the signals into detections.

[00:32:50] JM: And tell me about some of the hard engineering problems in building out that data pipeline.

[00:32:54] GP: Yeah. There are multiple parts of the hard engineering problem. I'll start with collect stage itself. For us, a couple of challenges that we encounter especially when it comes to our customer bases, fleet coverage. In the domain of security, usually the machines which are targeted in many cases are old machines for which the operating system is no longer supported by the vendor. The software patches may not be available. One of the nice things about osquery is that it's open source and you have the ability to compile it in such a way that it runs even on 15-year-old CentOS or Red Hat distribution for which the operating system vendor themselves might have stopped supporting the compilers and what have you.

So some challenges that we solve is ensuring that how do we get this executable so that you can get from 60 to, let's say, 98% fleet coverage when a fintech organization, as an example, might want to go across their 300,000 plus servers, right? So missing a good 30% plus implies that you're leaving like a big hole when it comes to like coverage. So that's one aspect of what we do, and our expertise from a system level allows us to like ensure that we can get it working on older versions. And then the second aspect is that we have to ensure that all of this is performant, because when you run this in environments, which are specifically cloud-centric environments, which is where the production of an organization is. Each time you eat into something, there's a challenge because you're taking up CPU resources, which would otherwise be used for production needs.

So another part that we've dealt with is how do we capture osquery telemetry at very large scale in the sense it's a vertical scaling problem for a given unit machine, but do it in a

performant way, but play shift all the analytics and send it to a backend. Now once you say that per unit endpoint you've figured out how and what it means to be performant and be efficient and connecting to a backend, the second challenge that we had to figure out was how do we scale the backend? Just scaling connectivity-wise, there are popular techniques especially given our background and having come from Akamai and in the past having built networks, which scale for content delivery and call processing needs, we were of course fortunate to pick some of well-established principles like using consistent hashing and other approaches to build a scalable detection network, which we call as an extensible detection network at the backend.

But one of the challenges that we had to face was what I mentioned a little bit earlier. When you say that it's a network of endpoints, given the problem of if you see anomalous behavior on one machine or outlier behavior, how do you turn it around and have quick access to the remaining machines? Whether it's 9999 or for 10,000 system cohort, how do you quickly discern and identify who the rest are and how do you find the question out so that the response coming back allows the organizations the most important aspect of reduced meantime to detection? That's a big engineering challenge that we've solved when it comes to sending all of this out.

And then the last part was while there is plenty of examples in other domains, arguably we are among the thought leaders when it comes to applying streaming analytics at very large scale. To put things into perspective, when 300,000 servers in production are generating roughly around 200 megabytes plus of data per day, we see that if people decide to store all of this and retain it, it ends up being like around 330 terabytes of data a month. And if they decide to keep retention, it ends up being a petabyte worth of data that historically has to be looked back and correlated against what's happening right now.

Clearly this all gets into the realm of big data type challenges, genuinely big data is what I would argue. But more importantly, the combination of doing things while the data is in streams so that you're looking at it as transforming telemetry into signals. And then doing correlations based on history. There is a lot of precedence in terms of Lambda architectures

and prior approaches that people have brought to the table. But we've solved some of these problems at scale. I can say we've solved because we've got production proof today where we have clusters in production that's humming along really smoothly while monitoring well over 200,000 endpoints in production for large enterprises.

[00:37:55] JM: What else would you like to build into Uptycs?

[00:37:58] GP: At Uptycs, we have made a conscious choice that what we provide is going to be relevant to a significant percentage of an organization who is a part of our customer base. As opposed to trying to ingest wide variety of telemetry, we've zoomed-in and focused on what we think is most relevant to our customers. What that meant is that we started by providing visibility into operating system instances for productivity reasons, as in Mac and a Windows laptop. On the production side, we started providing visibility for both compliance as well as detection reasons in virtual machines containerized environment.

To this, we have now added capabilities so that we can also capture where the container as environments run, as in are they orchestrated by Kubernetes or by OpenShift, or if these are running in the context of some kind of a cloud service provider as in AWS and CGP? What this allows us to do on the production side is to get inside out visibility from the virtual machines and the containers coupled with outside invisibility from Kubernetes and what the cloud provider metadata is telling it. Very specifically cloud provider metadata, you can think of it as configuration, cloud trail activity and flow logs combined with what we're seeing with the osquery type agent from inside the virtual machine, right? So that addresses a broad variety of use cases on the cloud side.

Now in a similar way when we started off providing visibility into the operating system endpoint, of course our customer base staying true to the feedback that we got along coupled with their ideation is how are people actually using various cloud services and where exactly is their productivity happening? Not surprisingly, a lot of people are going to be using Office 365 and G Suite, and all these deployments provide sufficient audit data much like how cloud providers provide data for AWS and GCP. And if you're able to combine that data and provide

visibility for an organization to say what's happening on their productivity endpoints, as in Mac and Windows, I think that provides a very strong foundation for our customer base to say what's happening across their fleet of machines today, right?

So the key point that I want to emphasize is that when we provide visibility, we go very deep both into the agent as well as the backend, which can actually interpret the telemetry in a structured manner and provide a very high-fidelity and at scale security use cases that can be addressed using our platform. That's what we are up to, a limited set of agents for a limited set of operating systems and environment, but the combination of that with deep visibility we provide is where we are heading organizationally at Uptycs. And I think that that's a sufficiently large customer base and market for us, but that focus is what's got our customers excited.

[00:41:25] JM: Okay, Ganesh. Well, that sounds like a good place to wrap up. Thanks for coming on the show. It's been a real pleasure talking to you.

[00:41:30] GP: Thank you for having me, Jeffrey. Bye-bye.

[END]