Interview with Travis McPeak

RIK FARROW

Travis works at Netflix on the Cloud Security team. He enjoys building automation to increase security while boosting developer productivity. Travis is a core developer of the Bandit and Repokid open source projects and has presented at security conferences, including BlackHat USA 2017, Enigma 2018, and re:Invent 2017. He currently serves as the Bay Area Open Web Application Security Project chapter leader. In previous roles he has served on the OpenStack Security team and as a founding member of the Cloud Foundry Security Team.
travis.mcpeak@gmail.com

Rik Farrow is the editor of *;login:*.
rik@usenix.org

I missed attending Enigma this year, so I started watching the online videos as soon as they became available. As always, I was doing more than just satisfying my own curiosity—I was also looking for talks that deserve a wider audience. I found several, asked the speakers, and Travis McPeak was the first to respond.

The technique of least privilege goes back to the dawn of computer security. First published in 1973, and presented at the fourth SOSP, Saltzer and Schroeder [1] laid out the ideas for granting only the level of privilege needed so that a particular application can function as intended.

Travis McPeak adds a new spin on this technique by applying it to applications deployed in the AWS cloud, starting with a safe list of permissions and automatically removing unused permissions over time, using an application he shares on GitHub.

Rik Farrow: What is least privilege, and why is it so hard for developers to get right?

Travis McPeak: Least privilege is a classic case of a simple concept that is very difficult to apply at scale. The idea itself is intuitive: we should only give applications the permissions that are required to function correctly. This is useful because in the case that an application becomes compromised, we can constrain the potential impact. For example, we protect files on Linux systems by granting read/write/execute permissions to only the user or group that needs access. This protects application resources from other processes on the system that we may not trust. Browsers enforce tab-level isolation so that a compromised tab can't affect the confidentiality or integrity of other tabs. One important point about least privilege is that it's a moving target. As an application changes, permissions may need to be added or removed to match the new requirements. This is analogous to applications that require and then shed root privileges when no longer needed.

The problems with manually applying least privilege become increasingly apparent when an organization grows in size and complexity. If I work by myself on an application, it's pretty easy for me to add permissions as I need them. Removing permissions when they are no longer required is a bit more challenging as there is no built-in reminder or incentive to remove them. I can set periodic reminders for myself and remove things that aren't needed anymore, but this may not be my top priority. However, what about organizations with dozens of applications and hundreds of developers continuously developing and redeploying? Who is responsible for periodically cleaning up permissions?

In many organizations the security team is responsible for granting and revoking privileges. There are a few problems with this. The security team doesn't work on all of these applications, so they don't know when an application changed and no longer needs some of the permissions. Manual security reviews are also a big problem because security teams are trying to balance lots of high priority work with a relatively small number of staff. If security teams are expected to stay on top of application changes and manually adjust permission sets to least privilege, they may not have enough time left to perform other critical work such as

applying patches, building tools to support secure development, and reviewing applications for security vulnerabilities.

For these reasons security teams seem forced into a binary decision: if the application is "important" enough, manually review it, otherwise ignore it. Every application that is "important" and manually reviewed saps time from both the application's developers and the security team. Every application that's ignored presents a risk to the business.

RF: In your Enigma '18 talk [2], you mention an AWS mechanism for controlling privilege. Could you describe that mechanism?

TM: At Netflix we rely on Amazon Web Services (AWS) heavily as our cloud provider. AWS provides a powerful access-control system called Identity and Access Management (IAM) that gives us very fine-grained control over specific actions and the resources they apply to. Here's a simple example policy:

```
{
 "Effect": "Allow",
 "Action": ["s3:GetObject", "s3:PutObject"],
 "Resource": "arn:aws:s3:::example_bucket/example_path"
}
```

This policy statement grants read and write access to the "example_path" in the S3 "example_bucket." While this policy is simple, it can quickly become difficult to determine which actions and resources should be allowed in a policy. With thousands of permissions, AWS can be configured very granularly compared to Linux file permissions. This configurability makes it both a powerful tool for security teams and very complicated for regular users. Even IAM experts may have difficulty determining exactly which permissions are needed to support a given application workflow.

RF: What techniques did the security team at Netflix come up with to deal with maintaining or improving least privilege in their applications?

TM: We use data about the permissions and resources that are actually used by an application to remove permissions that aren't required. To understand how this works in our environment it is useful to track the life cycle of an application and its permissions, beginning with how it gets permissions in the first place. Rather than wasting the valuable time of both developers and the security team, we automatically grant, by default, most of the benign permissions that applications need to perform common tasks. When a developer creates a new application, an application-specific role with the default permissions is created on their behalf by our deployment tool. If the application needs to perform any unusual or potentially dangerous actions, the security team and developers will perform a manual review, but in most cases the default permissions allow the application to do everything it needs.

After the application has been launched, we begin profiling it with tools that collect the data that AWS provides about which permissions and resources are actually used. Once a threshold of time has passed, unused permissions are automatically removed. Our open-source tool Repokid [3] automatically calculates new policies that preserve used permissions, removes unused permissions, and rewrites the new policy over the old. This approach eventually generates perfect least-privilege policies because anything that is kept is, by definition, actually used by the application.

If developers require a new permission or need something that was previously taken away added back, the security team manually adds it, but this is done with a quick conversation rather than the manual security reviews that we used to have to perform. The reasoning is simple: if developers are asking for a permission, then they either need it and will use it, or it will be automatically removed.

Once an application stops being used entirely, Repokid will remove all of the permissions, and the role becomes powerless. This is important because unused and unmaintained applications are huge headaches for security teams. The old applications have the same permissions with which they were originally deployed but aren't receiving patches or attention. By automatically removing permissions from these unmaintained applications, we can close a huge security hole.

RF: What are the challenges to this approach and how do you address them in your solution?

TM: Some applications don't regularly use their permissions but need them on an infrequent basis. Our usage-based analysis fails for these applications, and if we aren't careful, we can break them. For this reason, it's important to identify such applications and exclude them. Fortunately, there are relatively few, and we can fall back to the traditional manual review process for them.

Another concern is the eventuality that we will break something. We have put a lot of thought and consideration into how we can recover quickly and also detect as rapidly as possible that we have broken an application. Our goal is to use high quality data sources to avoid breaking applications. If we do break something we detect it, and when we detect a broken application we can fix it with the push of a button by rolling back to an earlier permissions state.

At Netflix we are focusing on logistics such as how to inform developers when changes are occurring for their applications and to give them options to defer or entirely block changes. Our goal is for developers to view this as a service that the security team provides for them to automatically make their applications more secure. The better we can communicate this message, the more successful our program will ultimately be.

RF: Does this approach only work for AWS or can it be applied to other areas of security?

TM: The solution we developed is for AWS applications and permissions, but we think this approach extends equally well to other areas of security. For example, it should be possible to use the same kind of data to constrain application container permissions to allow only the required syscalls and capabilities. Applications running directly on Linux systems may similarly be constrained by AppArmor or seccomp profiles that are generated automatically based on usage profiling. Mobile application privileges may be reduced by profiling usage in a sandbox environment and then having required permissions suggested, taking the guesswork out of permissions for the application's developer. There are many other possibilities for a similar approach applied to other areas of security in the future.

RF: What are the next steps for the project and its application at Netflix?

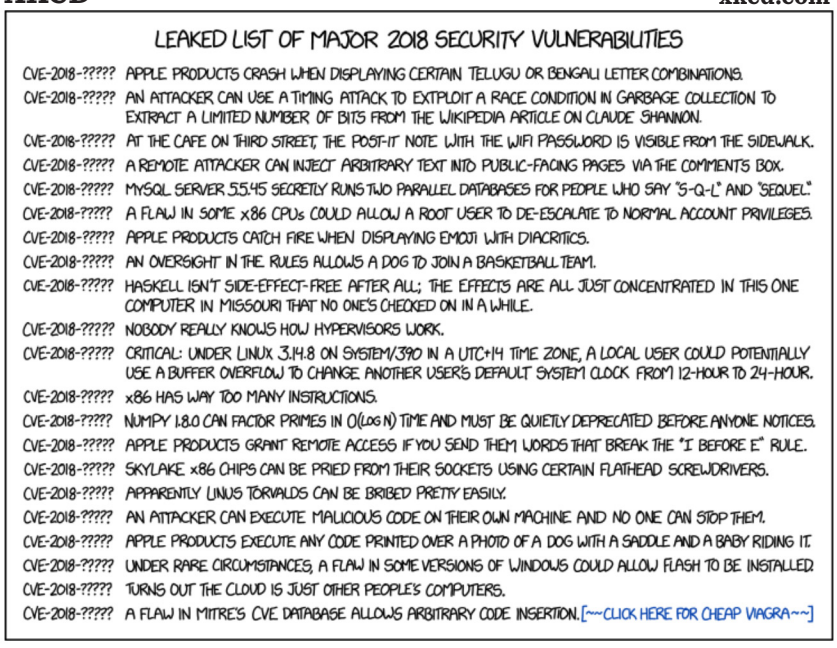
TM: We are continuing active development on the Repokid [3] project. As we add more data sources, we'll gain both more confidence in the policy suggestions and the ability to trim unused permissions even further. Specifically, we're excited about using data to constrain policy access to resources. We're starting with S3 but look forward to protecting other resources as well. Another data source that might be interesting is the software we're deploying itself. By examining the package, we may be able to infer what access it needs and remove access that it doesn't. As always, we welcome contributions to Repokid and feedback from organizations that are using it.

References

[1] J. Saltzer, M. Schroeder, "The Protection of Information in Computer Systems," in Fourth ACM Symposium on Operating System Principles (October 1973): http://www.cs.virginia.edu/~evans/cs551/saltzer/.

[2] T. McPeak, "Least Privilege: Security Gain without Developer Pain," ENIGMA Conference (USENIX, 2018): https://www.usenix.org/conference/enigma2018/presentation/mcpeak.

[3] Repokid: https://github.com/Netflix/repokid.



ENIGMA

A USENIX CONFERENCE

Submit a Talk

Enigma centers on a single track of engaging talks covering a wide range of topics in security and privacy. Our goal is to clearly explain emerging threats and defenses in the growing intersection of society and technology, and to foster an intelligent and informed conversation within the community and the world. We view diversity as a key enabler for this goal and actively work to ensure that the Enigma community encourages and welcomes participation from all employment sectors, racial and ethnic backgrounds, nationalities, and genders.

Enigma is committed to fostering an open, collaborative, and respectful environment. Enigma and USENIX are also dedicated to open science and open conversations, and all talk media is available to the public after the conference.

PROGRAM CO-CHAIRS

Ben Adida
Clever

Franziska Roesner,
University of Washington

The submission deadline is August 22, 2018.

enigma.usenix.org

JAN 28–30, 2019

BURLINGAME, CA, USA

usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION