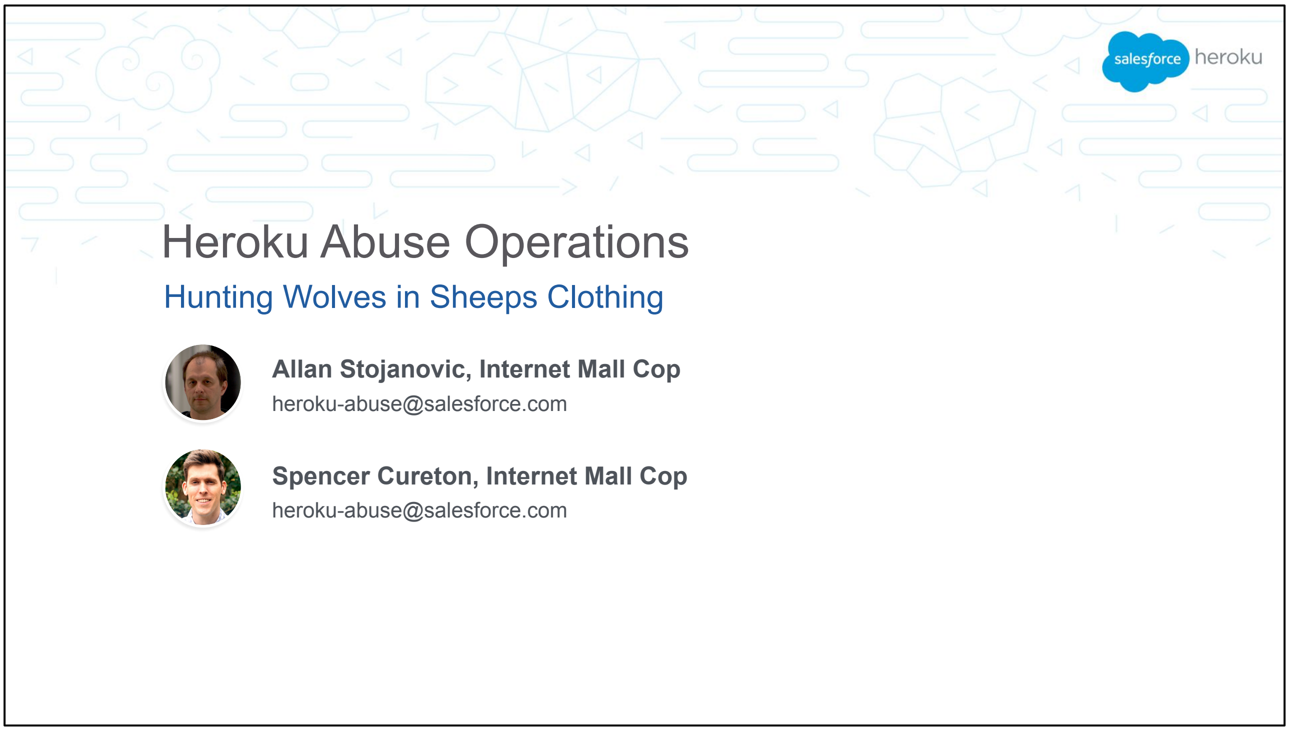
black hat
USA 2020
AUGUST 5-6, 2020
BRIEFINGS

Heroku Abuse Operations:

Hunting Wolves in Sheeps Clothing

#BHUSA @BLACKHATEVENTS

Forward-Looking Statement



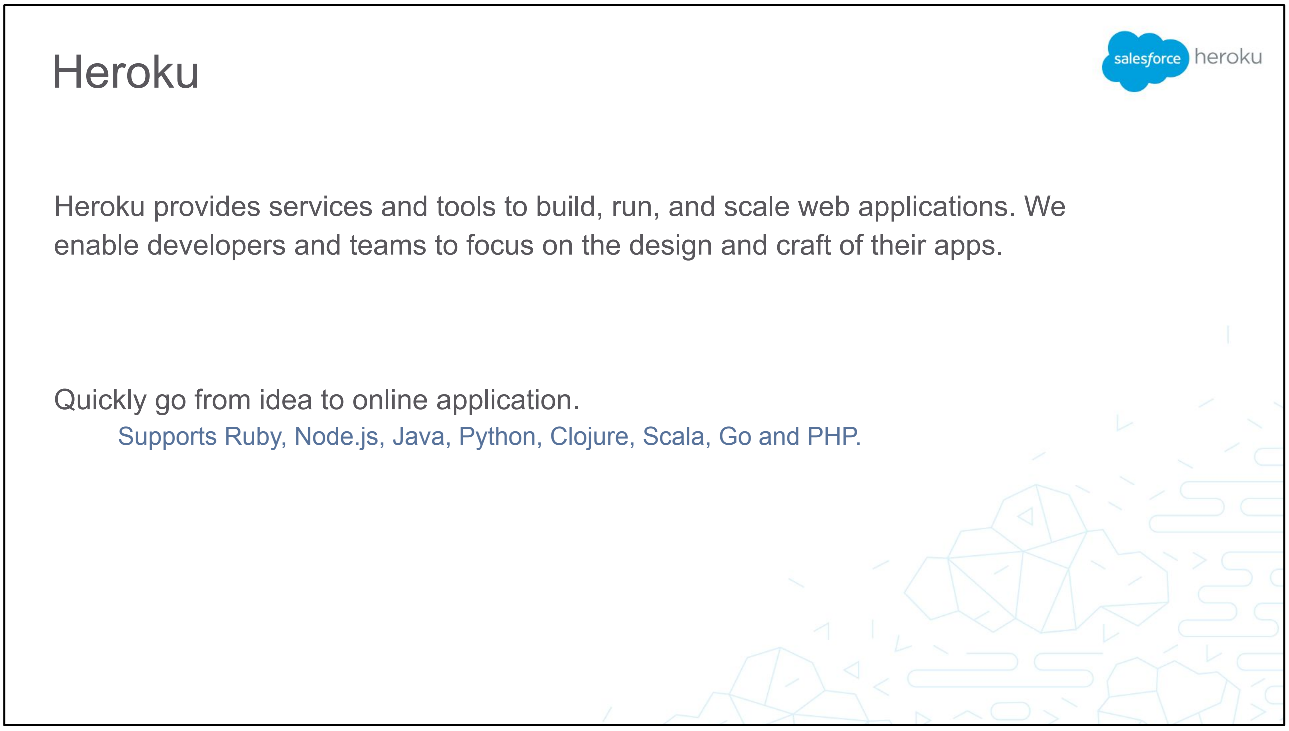
salesforce heroku

Heroku Abuse Operations

Hunting Wolves in Sheeps Clothing

Allan Stojanovic, Internet Mall Cop
heroku-abuse@salesforce.com

Spencer Cureton, Internet Mall Cop
heroku-abuse@salesforce.com



Heroku

Heroku provides services and tools to build, run, and scale web applications. We enable developers and teams to focus on the design and craft of their apps.

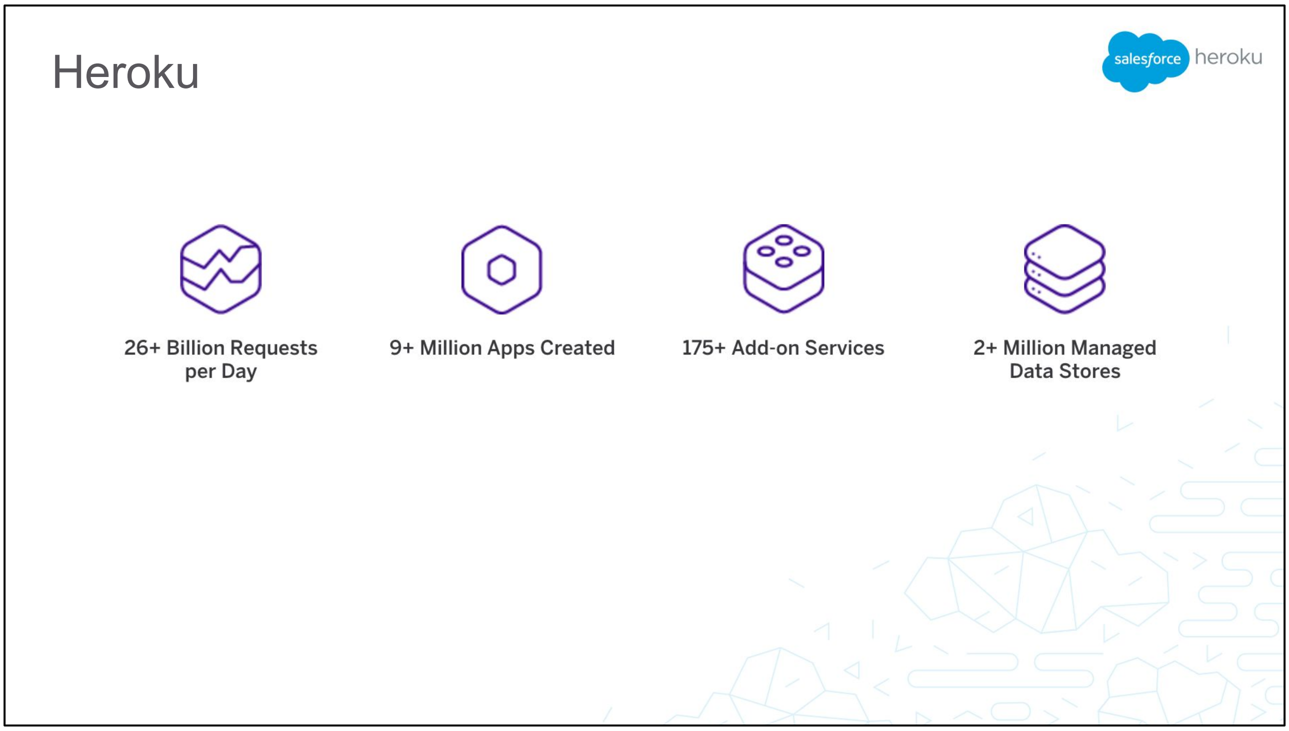
Quickly go from idea to online application.

- Supports Ruby, Node.js, Java, Python, Clojure, Scala, Go and PHP.

Heroku was purchased by Salesforce about ten years ago.

It’s a platform as a service that offers an easy interface to quickly go from idea to online application without having to manage and configure the underlying infrastructure. Developers are free to focus on code and deploying it to the internet without much effort.

It supports multiple languages and provides features like scalability, backups, continuous integration, sane default configuration and much much more. And all you need is an email address to create an account.



Heroku

salesforce heroku

26+ Billion Requests per Day

9+ Million Apps Created

175+ Add-on Services

2+ Million Managed Data Stores

And to give an idea about the size of the platform - over 9 million apps have been created, over 2 million databases are managed and the infrastructure handles billions of requests per day.

Oh, and Heroku runs on Heroku.

Heroku

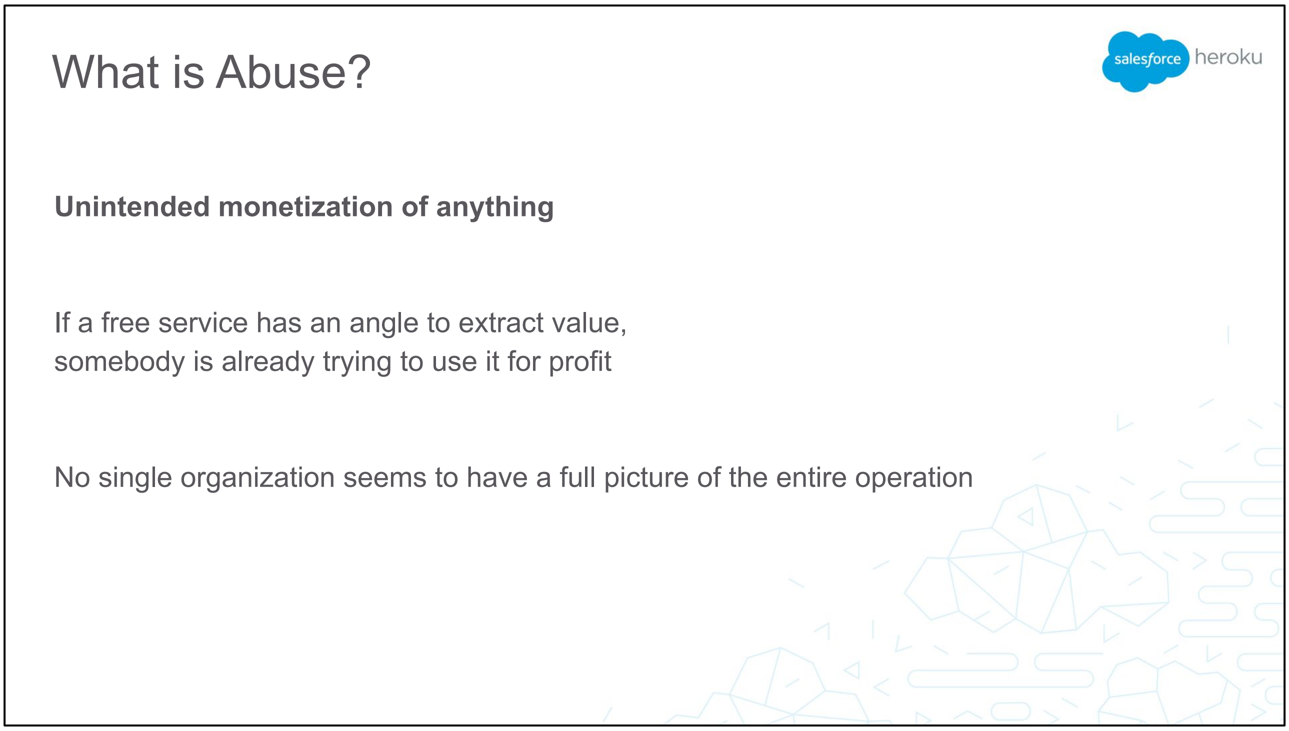
~~Platform as a Service~~

Remote Code Execution as a Service

PaaS…
More like RCEaaS

Heroku has other offerings that are not like this, but we won’t be talking about them today.

Our focus will be on how we defend the “common runtime” from abusive bad actors.



What is Abuse?

Unintended monetization of anything

If a free service has an angle to extract value,
somebody is already trying to use it for profit

No single organization seems to have a full picture of the entire operation

We define “abuse” as unintended monetization of anything.

If you have a free service, a free trial, or any offering that has an angle to extract value, somebody is already trying to figure out a way to make money off of it.

It’s a very fine line between a business trying to get off the ground and an abuser trying to take **advantage** of your services. Sometimes it’s obvious - theft of resources. Other times it’s not so clear.

For example, a social media poll. Intended for honest input, but gameable because it’s not one person - one vote. It’s one *account* - one vote. The person in control of the most accounts wins.

The problem isn’t new, the scams aren’t new, but we see them leveraging more of the cloud.

Is there a single view that captures the full picture of the entire operation?

Across the industry, we may have that full picture, but not right now.
From our limited view, we’ve identified a number of puzzle pieces that have been underreported.

What is Abuse?

Misuse, malice and crime on the platform

Now on a more practical layer, we see a portion of that picture realized through misuse, malice and crime on our platform.

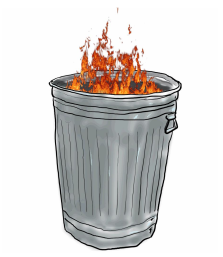
Legal documents like the Acceptable Use Policy, the AUP, provide a list of prohibited content and actions - the list of “don’t’s” for users to follow, as well a set of guidelines for us to uphold.

Obviously, this can’t cover every type of abuse activity. Even if it did, it would act as a laundry list for an abuser to know what badness _can_ be performed from the platform.

Let’s look at how we group AUP violations.

AUP Violators

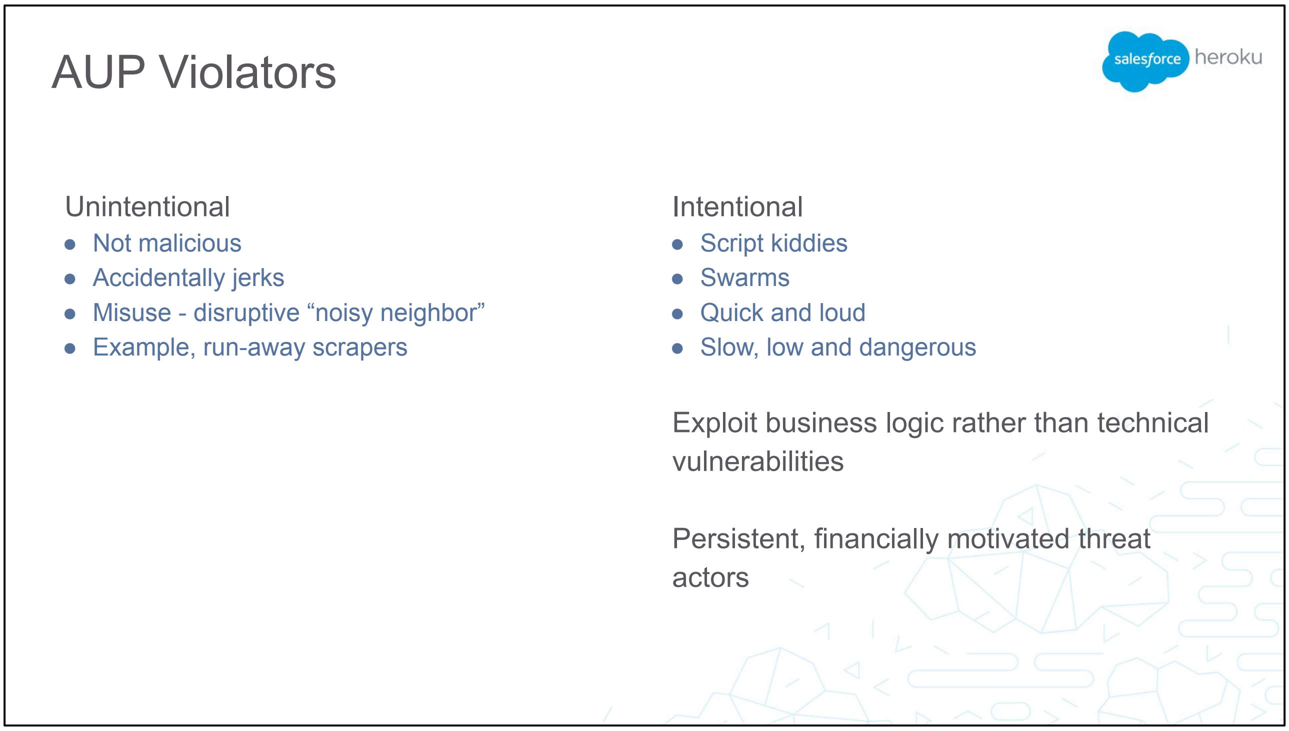
Unintentional



Intentional

There are unintentional violations and intentional violations

To determine if intentional or unintentional, good or bad, we establish intent by collecting and connecting as many indicators of abuse as necessary.



AUP Violators

Unintentional
- Not malicious
- Accidentally jerks
- Misuse - disruptive “noisy neighbor”
- Example, run-away scrapers

Intentional
- Script kiddies
- Swarms
- Quick and loud
- Slow, low and dangerous

Exploit business logic rather than technical vulnerabilities

Persistent, financially motivated threat actors

Unintentional could be a customer that deploys a poorly written app that consumes more than its share of resources and now they’ve become a “noisy neighbor”. Not malicious, but kind of like they were accidentally jerks, totally unaware until we suspend their app and open a support ticket.

The intentional abusers are what make this job really interesting.
Users with skills that range from script kiddies to rogue, advanced developers, which we call our “apex actors”.

Skids will be skids and deploy code found on github, youtube, or whatever their buddy gave them to run.

At times, we’ll get what feels like a swarm of abuse centered around a particular app, like a proxy kit or a P2P app.

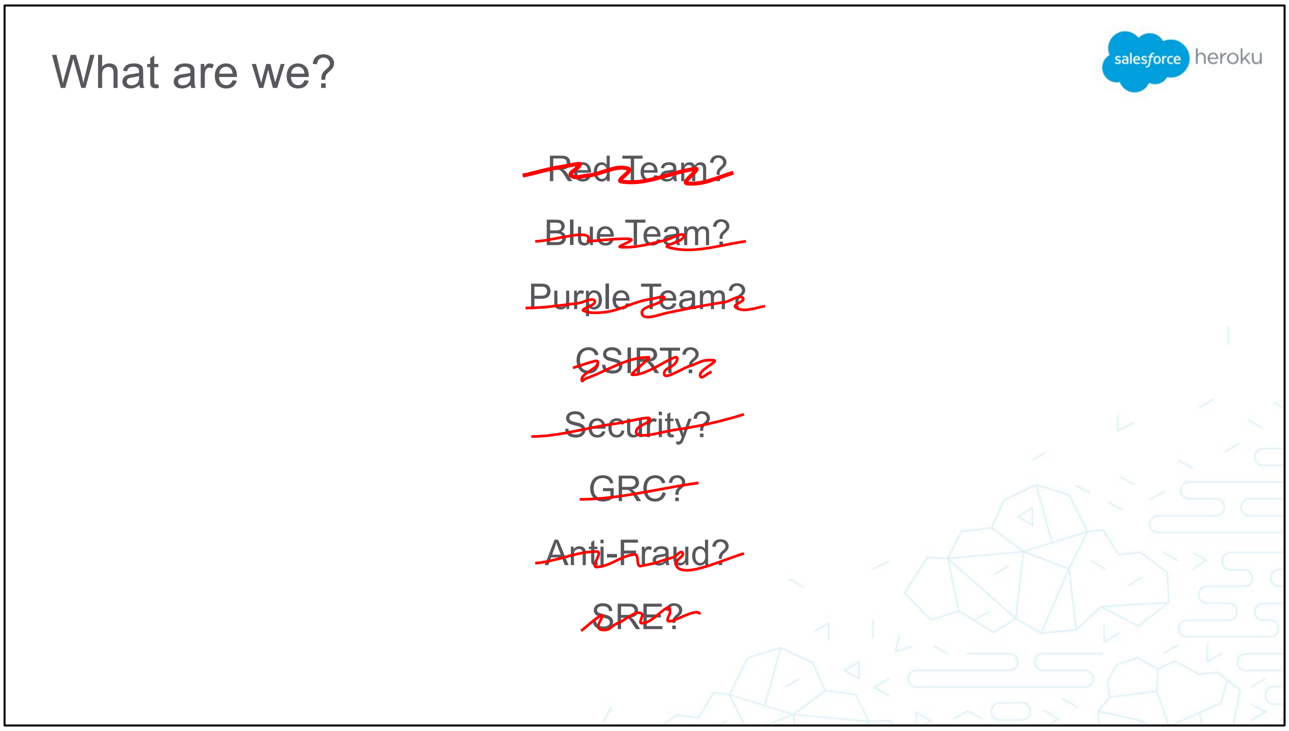
Then there are the “quick and loud” abusers that could care less about being detected and will attempt to stand-up a DDoS farm and attack a target as quickly as possible.

The actors that are patient, skilled and criminal are the more dangerous group. These actors attempt to iterate past our detection capabilities, they may social engineer our support staff in order to gain insight into how they were detected after being suspended, and will try anything to maximize their use of our platform.

Now, these users are not exploiting technical vulnerabilities in the platform. They take advantage of **gaps in business logic**.

The majority are not too advanced, but are very persistent. Think APT with little “a” and big “P” and again, financially motivated.

And by “persistent” we mean that after an account has been suspended, the same actor will use a different email address or a subinbox of the original email address to create a new user account.



What are we?

~~Red Team?~~

~~Blue Team?~~

~~Purple Team?~~

~~CSIRT?~~

~~Security?~~

~~GRC?~~

~~Anti-Fraud?~~

~~SRE?~~

Here's a fun one - how do we define ourselves? Where do we fit within the company?

Let's start by listing the things that we're not.

We're
Not Red Team, but we learn from them
Not Blue Team, but we borrow from them
Not purple team, but we talk to more groups across the organization
Not CSIRT, but we conduct investigations
Not Security, but we monitor for outbound attacks
Not GRC, but we write up risks around business rules
Not Anti-Fraud, but use the same data
Not service reliability, but we help keep the platform stable.

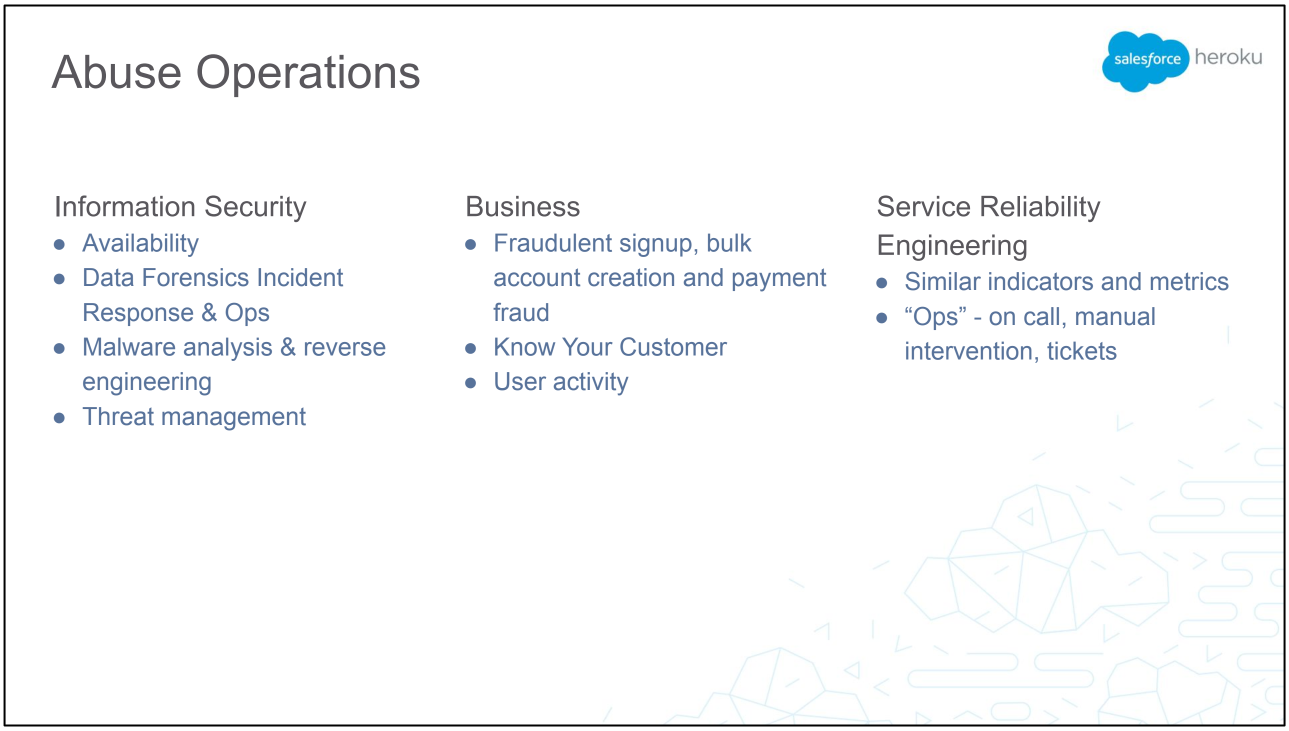
We're not encompassed by any one of these groups, but we overlap in so many ways.

What are we?

Abuse Operations

We're Abuse Operations.

We exist because the business understands that there is risk of having unwanted activity given an environment that is RCE as a service.



Abuse Operations

Information Security

- Availability
- Data Forensics Incident Response & Ops
- Malware analysis & reverse engineering
- Threat management

Business

- Fraudulent signup, bulk account creation and payment fraud
- Know Your Customer
- User activity

Service Reliability Engineering

- Similar indicators and metrics
- “Ops” - on call, manual intervention, tickets

Overall, we define Abuse Operations as a specialized DFIR team that requires a wide view across the business, as shown.

Our goal is to push bad actors from the platform and put a spotlight on abuse patterns for engineering teams and the business to address.

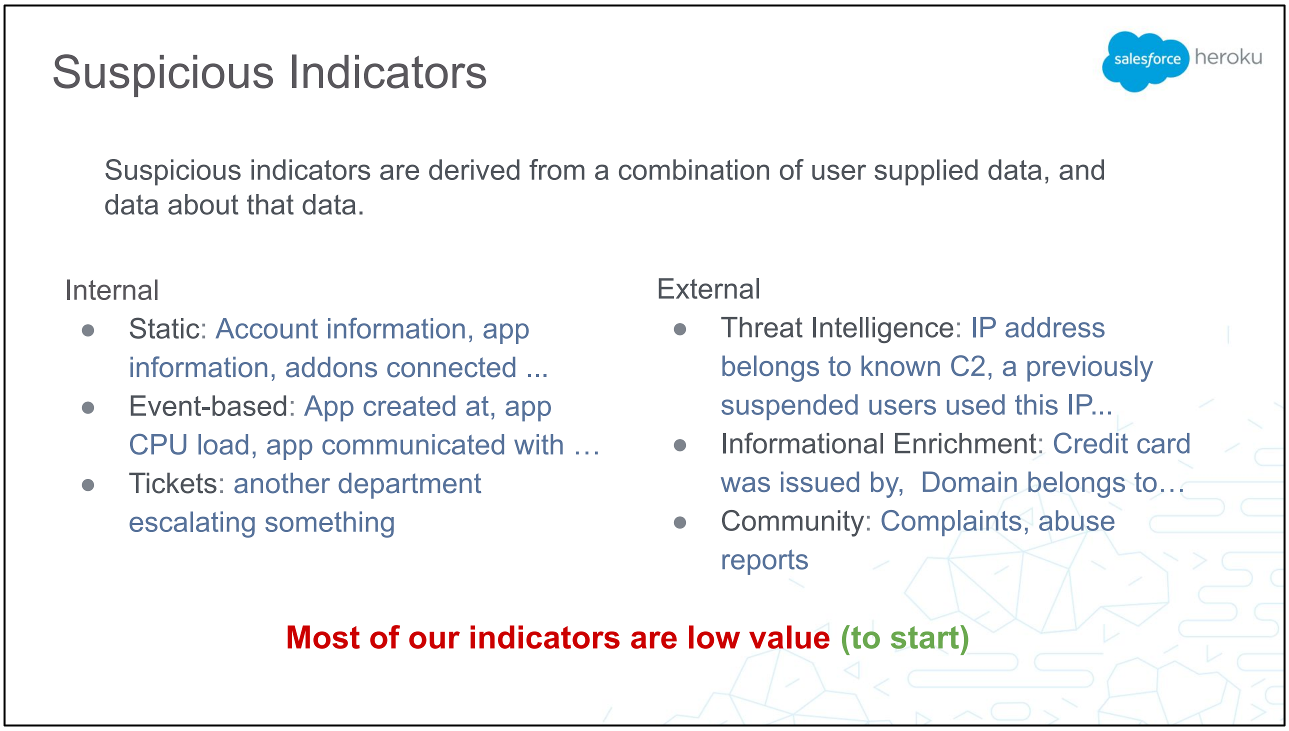
Heroku is exceptionally collaborative, where teams openly share views and metrics and ping us when things look odd on their end. This enables us to move very fast.

Hunting For Badness

We. Hunt.

We don’t want to wait for the badness to hurt us. We are looking for wolves in sheep’s clothing, after all.

Most of our hunting starts with ...



Suspicious Indicators

Suspicious indicators are derived from a combination of user supplied data, and data about that data.

Internal

- Static: Account information, app information, addons connected ...
- Event-based: App created at, app CPU load, app communicated with …
- Tickets: another department escalating something

External

- Threat Intelligence: IP address belongs to known C2, a previously suspended users used this IP...
- Informational Enrichment: Credit card was issued by, Domain belongs to…
- Community: Complaints, abuse reports

Most of our indicators are low value (to start)

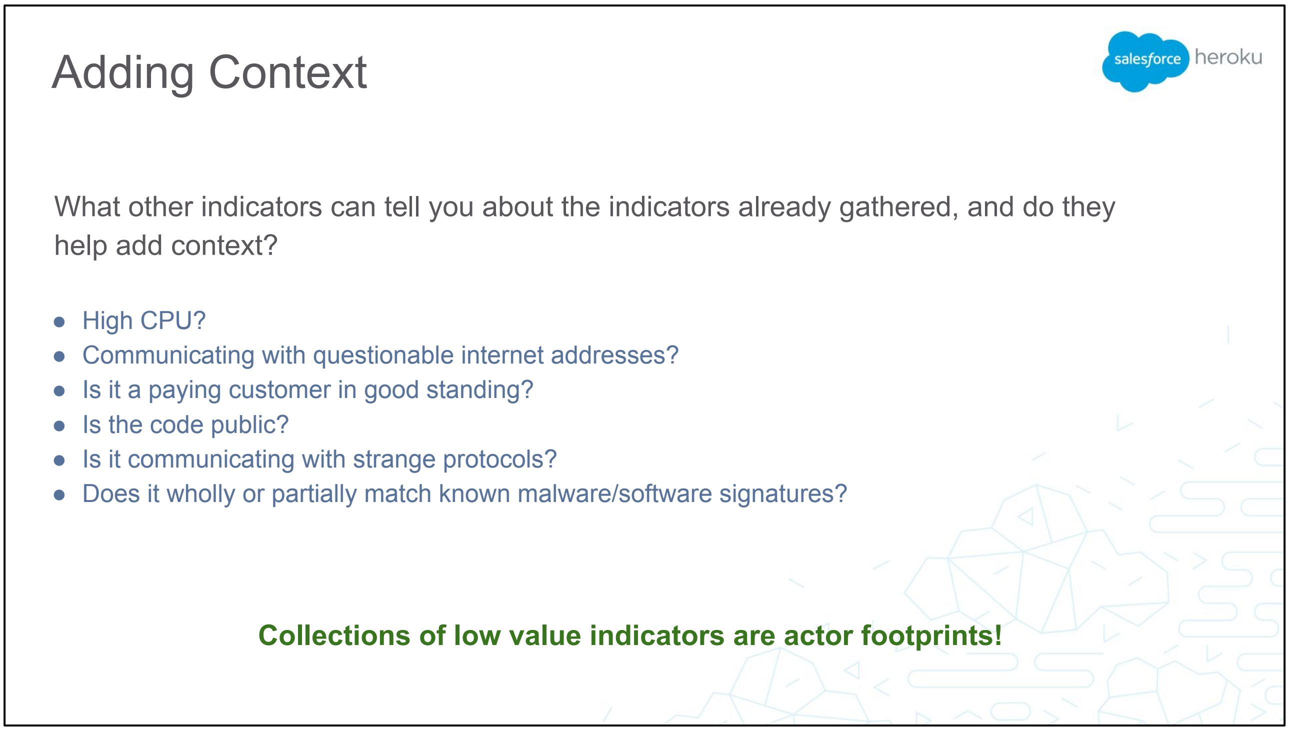
The hunt starts with a suspicious indicator. Suspicious indicators are derived from a combination of internal user data, and external information about that data.

For example, an app with an abnormally high CPU load would generate an event based alarm (internal indicator).

Internal indicators point out odd or unusual activity that should be investigated. They can come from any of a number of sources, but usually though some sort of alert derived from user data or platform activity.

External indicators are usually derived from complaints, abuse reports, threat intel, and Informational enrichment.

Individually these indicators are of low value because they are _not atomic_. If this doesn’t make sense yet, don’t worry, we’ll get there.



Adding Context

What other indicators can tell you about the indicators already gathered, and do they help add context?

- High CPU?
- Communicating with questionable internet addresses?
- Is it a paying customer in good standing?
- Is the code public?
- Is it communicating with strange protocols?
- Does it wholly or partially match known malware/software signatures?

Collections of low value indicators are actor footprints!

Given a starting suspicious indicator or two, we form one or more hypothesis about the intent of the actor. We start looking for related indicators to try and prove or disprove that hypothesis.

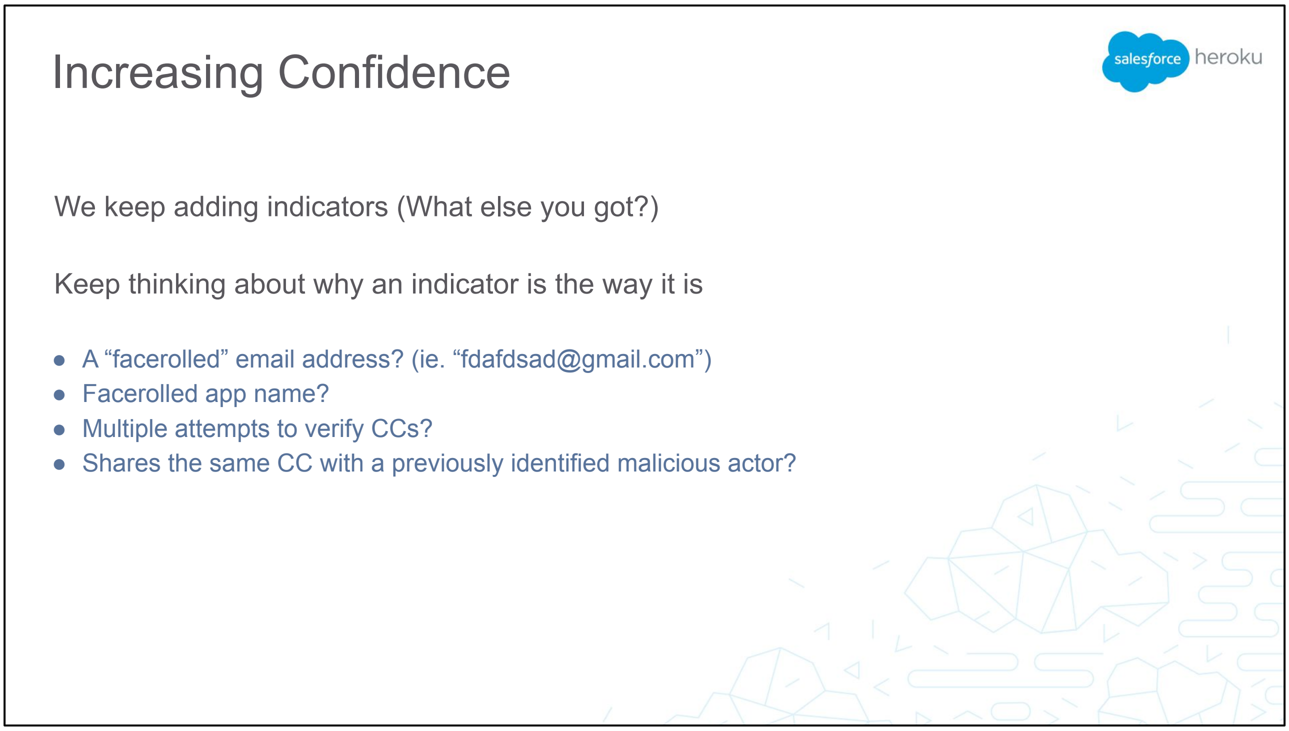
This is where it starts to get interesting.

For instance, it just so happens that given:

- High CPU
- High Network utilization
- Match for a popular file download program
- And the substring “torrentz” in the app name

… is probably a P2P app. True story.

Each of these indicators is not proof of intent individually, but you can see that collectively they make a very strong case. A collection of low value indicators does become very good footprint of malicious activity.



Increasing Confidence

We keep adding indicators (What else you got?)

Keep thinking about why an indicator is the way it is

- A “facerolled” email address? (ie. “fdafdsad@gmail.com”)
- Facerolled app name?
- Multiple attempts to verify CCs?
- Shares the same CC with a previously identified malicious actor?

To increase confidence, we keep asking the question, “What else ya got?”

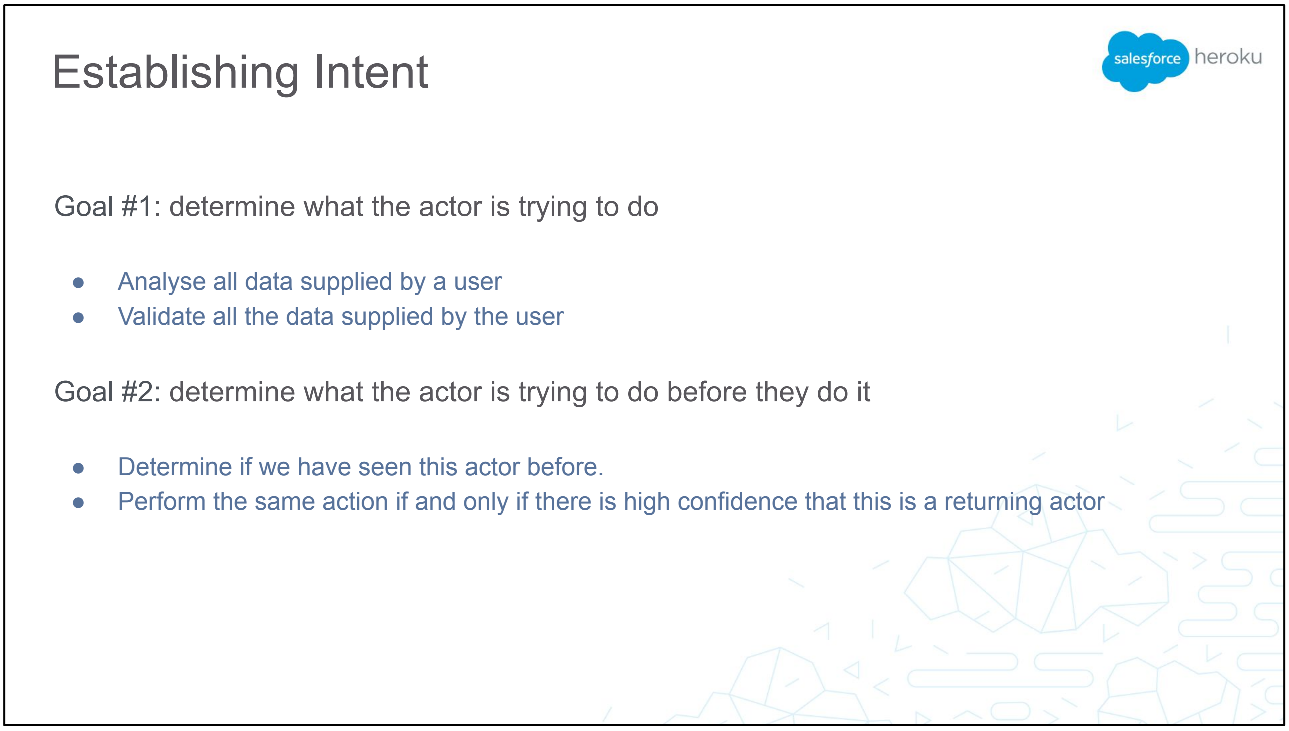
And as more indicators are discovered, we try and think about why they are the way they are.

For instance, a “facerolled” email address, like shown, is a pretty good indicator that the **user does not really value the account**. It is likely a throw-away account, this by itself is not necessarily bad.

A “facerolled” app name, similarly, is not necessarily bad, it could be something built for a quick test, meant to be discarded.

However, a facerolled email address with a collection of facerolled app names, _and_ multiple failed attempts to verify different credit cards, and finally landing on a working credit card that has been previously used by a previously identified malicious actor, is worthy of further scrutiny. Don’t you agree?

By the way, in case it is not clear, the difference between a “facerolled” string and a “random” string is whether it was created by hand or by computer. Think “Keyboard Mashing”



Establishing Intent

Goal #1: determine what the actor is trying to do

- Analyse all data supplied by a user
- Validate all the data supplied by the user

Goal #2: determine what the actor is trying to do before they do it

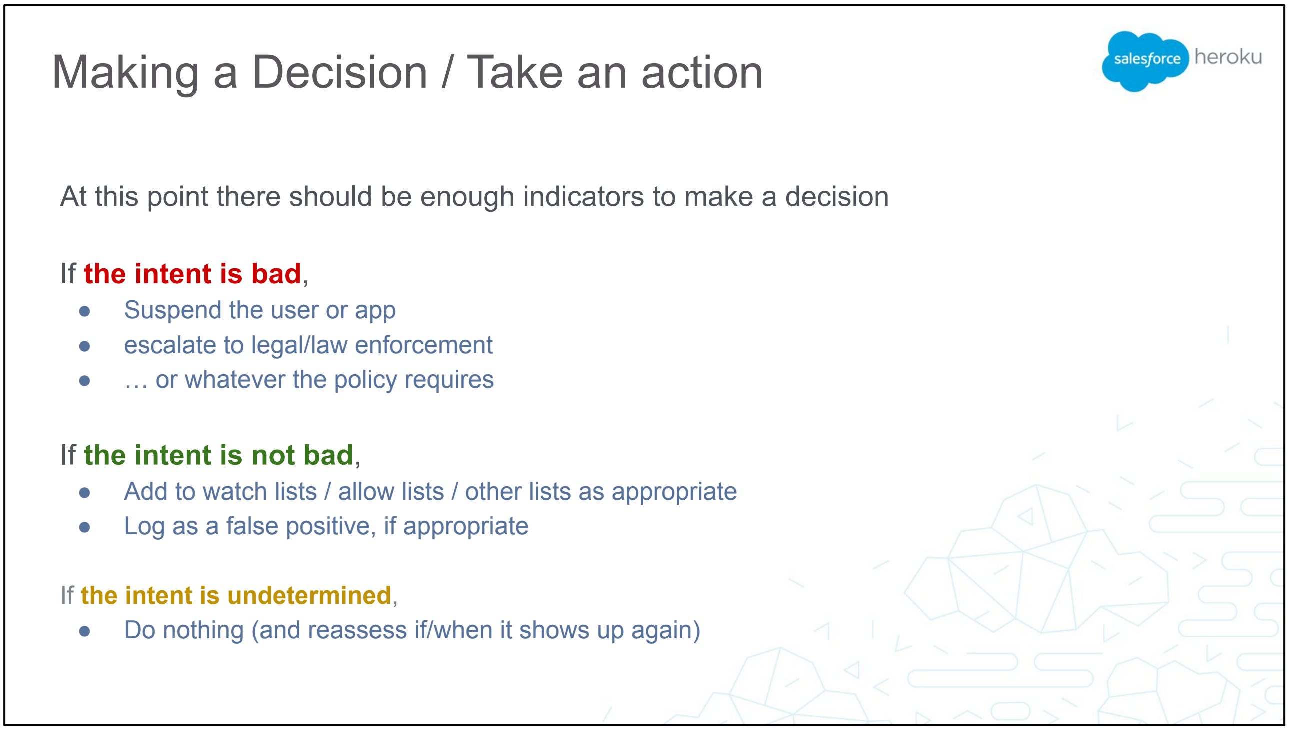
- Determine if we have seen this actor before.
- Perform the same action if and only if there is high confidence that this is a returning actor

Establishing intent is, as one would expect, the art of figuring out what the actor is trying to do.

Goal #1 is determining what the actor is trying to do

Goal #2 is determining what the actor is trying to do before they do it.

We reserve goal #2 for actors we have seen before to avoid the “precrime” scenario. We try really really hard not to kick someone off the platform before they have “done bad”.



Making a Decision / Take an action

At this point there should be enough indicators to make a decision

If **the intent is bad**,

- Suspend the user or app
- escalate to legal/law enforcement
- … or whatever the policy requires

If **the intent is not bad**,

- Add to watch lists / allow lists / other lists as appropriate
- Log as a false positive, if appropriate

If **the intent is undetermined**,

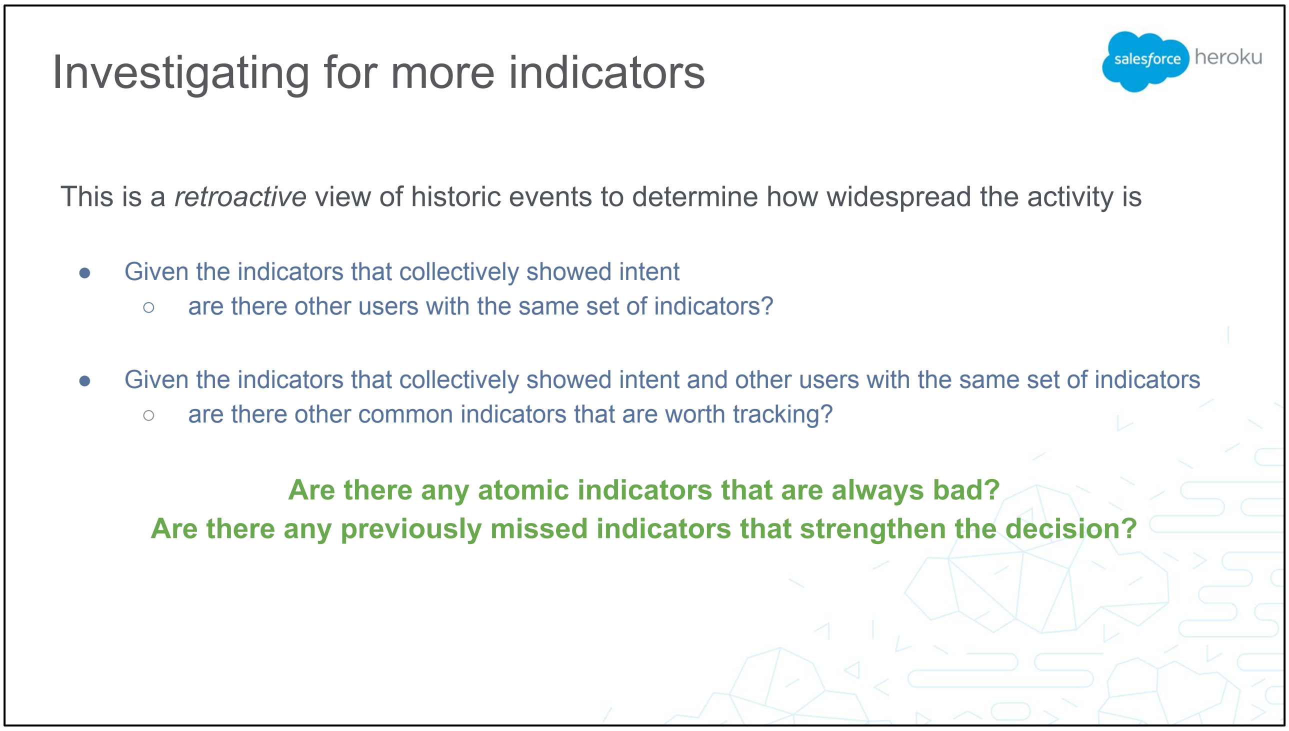
- Do nothing (and reassess if/when it shows up again)

Making a decision and taking an action is the crux of the matter. At this point, we have collected a bunch of indicators that should give us a reasonably good idea of the intent of the activity. It is time to take action.

If the intent is bad, then suspensions / evidence collection / deletions, etc. come into play.

If the intent is NOT bad then watchlists or allow lists, adjustments for false positives, and/or opening of support tickets are among the paths forward. Analyst discretion is necessary here; is it GOOD, or just not BAD yet?

Sometimes undetermined intent really just means that there are not enough indicators that help establish the intent of the activity. In which case it may go “back into the pool” as no other actions can be taken. It will be reassessed if something perculates it to the top again.



Investigating for more indicators

This is a *retroactive* view of historic events to determine how widespread the activity is

- Given the indicators that collectively showed intent
  - are there other users with the same set of indicators?
- Given the indicators that collectively showed intent and other users with the same set of indicators
  - are there other common indicators that are worth tracking?

Are there any atomic indicators that are always bad?
Are there any previously missed indicators that strengthen the decision?

We have a bunch of indicators. We have enough to estable intent. We made our decision, and we've taken the corresponding action. Now, we look to see how widespread this activity is.

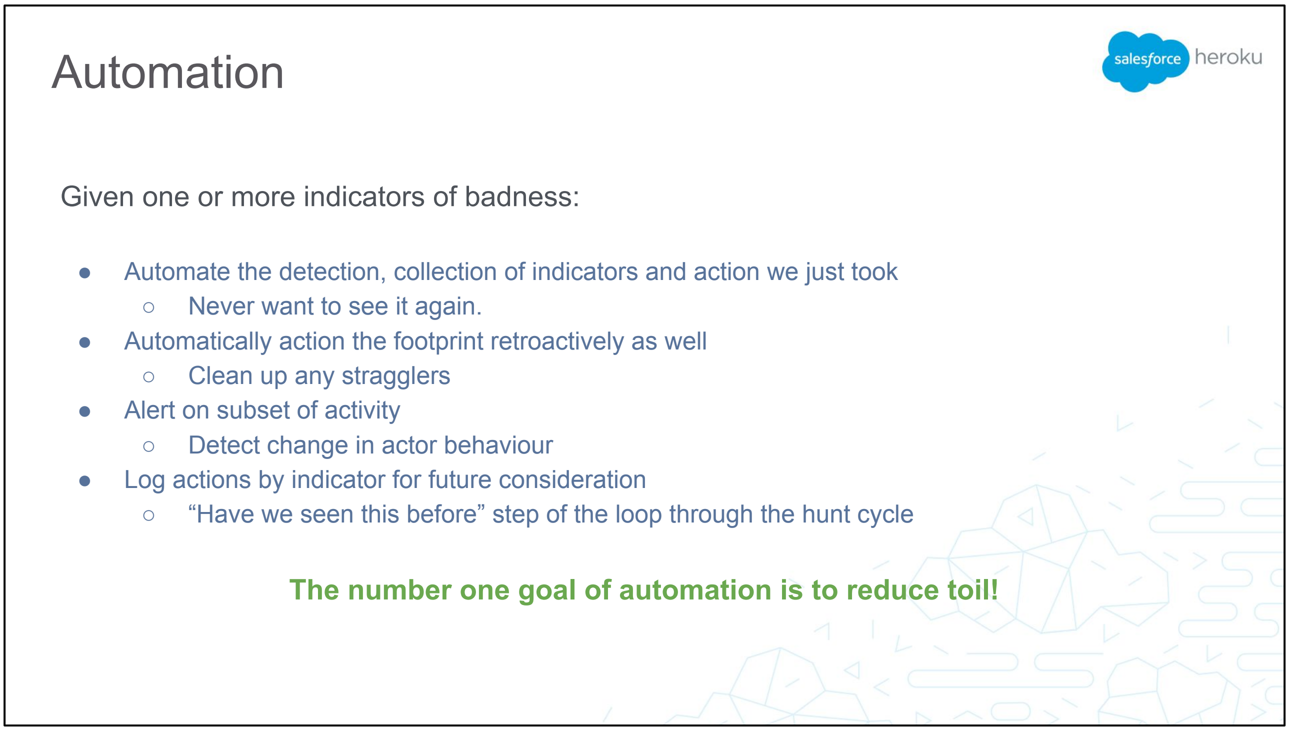
We gather all the accounts associated with those indicators we have, and look for other common indicators that may have been missed. We then add those indicators to our collection, and see if any accounts match that we may have missed.

If one gets lucky, there may be a single very strong indicator (what we call an atomic indicator) that ties all the accounts or apps together and cannot be interpreted any other way than being an indicator of Bad Intent.

If not, the entire collection of indicators already gathered can be augmented with the new set, groomed, and listed as the new footprint of badness.

Sampling across a larger population of users or apps, will quickly confirm the accuracy of the footprint, and help weed out any remaining false positives.

This strengthens the footprint and makes it suitable for …



Automation

Given one or more indicators of badness:

- Automate the detection, collection of indicators and action we just took
  - Never want to see it again.
- Automatically action the footprint retroactively as well
  - Clean up any stragglers
- Alert on subset of activity
  - Detect change in actor behaviour
- Log actions by indicator for future consideration
  - “Have we seen this before” step of the loop through the hunt cycle

The number one goal of automation is to reduce toil!

Automation is how we reduce toil. We never want to see the same abuse again, so we automate the collection, detection and actioning of the footprint we just actioned manually.

Our automation can also be set up to run the new footprint retroactively, so that older implementation of badness will also be detected and actioned as well. This cleans up any stragglers.

We log the actions taken so that we can use that information the next time we run though the loop: this is how we get the “have we seen this before” question answered.

There is a trade off. Automation will result in some loss of visibility. The investigation performed in the previous step helps build robustness so that there is longevity in the automation of this action. Counterintuitively, taking that extra step helps reduce toil in the long run.

Why do we use the term footprint? Simply because fingerprint or profile implies something too unique. A footprint, as a collection of indicators, helps us remember that an actor can be more than one person, can be a bot, can be multiple specialised actor groups, or any combination thereof.

Rinse and Repeat

And then we do it all over again.

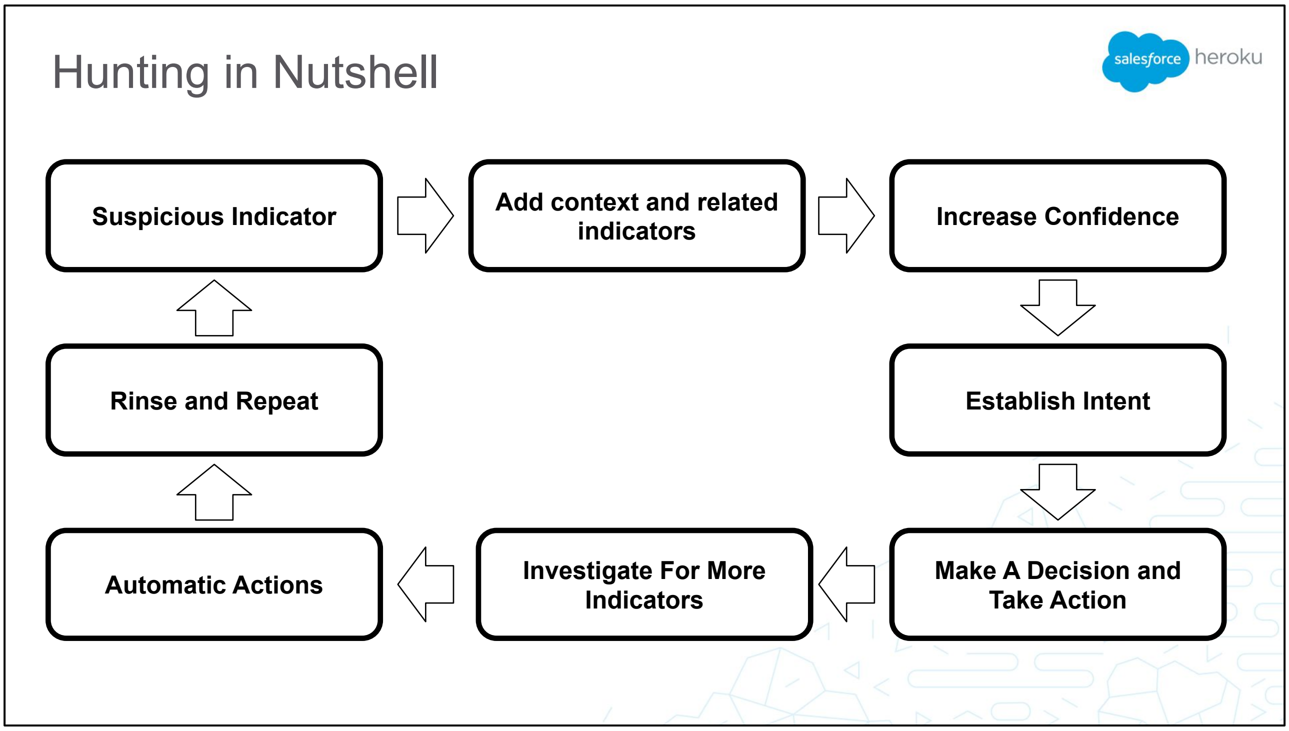
********************* extra info **************************

Logging every action that the system takes allows for analysis of the effectiveness of the system over time, and allows for the automation of reporting. Who doesn't love reporting?

We found that implementing partial footprint actions can be tricky to get right, so to avoid a runaway false positive streak, we implement new rules in an alert-only mode, monitor it until the analyst is satisfied, and then turn it to "action mode". We then continue to monitor it, and if it turns out that it is not correct, we can turn it back to an alert-only mode very rapidly.

We have a number of different allow-lists implemented to help avoid total disaster, certain VIP customers are on the list (especially those with their own security teams), as are certain technologies that we simply cannot put at risk.

Overall this path is a lot to take in. But if you have familiarity with network monitoring tools, malware fingerprinting, and / or basic behavioural analytics, you can see that we borrow heavily from these techniques, frameworks and more.



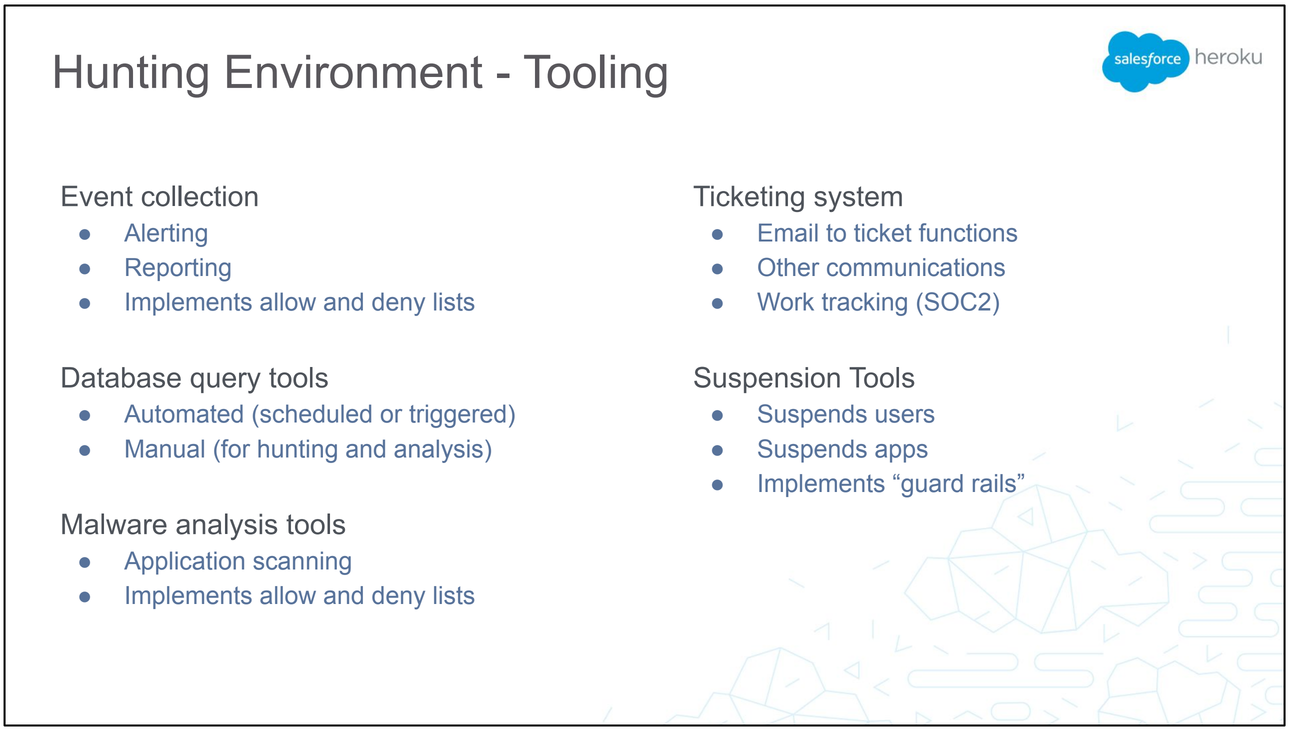
Hunting in Nutshell

To recap:
suspicious indicators come to us as high watermark alarms, abuse complaints or as a result of artifact exploration (including previous historic indicators)

We add context and more indicators from both internal and external sources ...
… to increase confidence...
… through establishing intent ...
… to make a decision.

We then investigate, often retroactively, to find more indicators that collectively are suitable for automation.

Rinse and repeat.



Hunting Environment - Tooling

Event collection
- Alerting
- Reporting
- Implements allow and deny lists

Database query tools
- Automated (scheduled or triggered)
- Manual (for hunting and analysis)

Malware analysis tools
- Application scanning
- Implements allow and deny lists

Ticketing system
- Email to ticket functions
- Other communications
- Work tracking (SOC2)

Suspension Tools
- Suspends users
- Suspends apps
- Implements “guard rails”

We have an event collector that acts as a classic SIEM. It handles our alerting and reporting, and can send info to other tools as well as receive.

We have database query tools that can run queries automatically, and we also run them manually when we test hypotheses, or results of automatic suspensions.

We have malware analysis tools that can scan apps statically and dynamically, and send results to other tools.

We have a ticket system, most places don’t consider ticket systems part of the tooling, but we do if for no other reason that it is responsible for a lot of our toil, so we need to be able to reduce toil there as well.

And we have our suspension tools that suspend users and apps. This also encompasses complex rules implemented as guard rails so that we don’t accidently suspend the world.

******************** extra info *************************

Event collection
Our SIEM install is big. When I first started the number banded about was 63 TB of logs per _day_. I do not know how big it is now, but I know it has grown. Back-of-the-napkin math has also guesstimated that if we actually gathered everything we wanted we would be adding another 20 PB of logs to the system per

day (yes, PetaBytes … with a P). This is not feasible for all the reasons. We are resigned to the fact that our SIEM has so much data, but never enough. Our SIEM instance is also shared across the entire organization. The good news there is that is why we have close proximity to performance and metrics data.

Some of our footprint tracking is implemented in the SIEM as reports and alerts. Things that we find interesting can kick off other processes using webhooks or email.

Database query tools

Our database tools are Heroku apps that can make automated database calls that look for footprints and take an action, including kick off other processes, and sending logs. We can also run many queries manually, and we often use that to test hypotheses and double check automated outcomes.

Malware Analysis

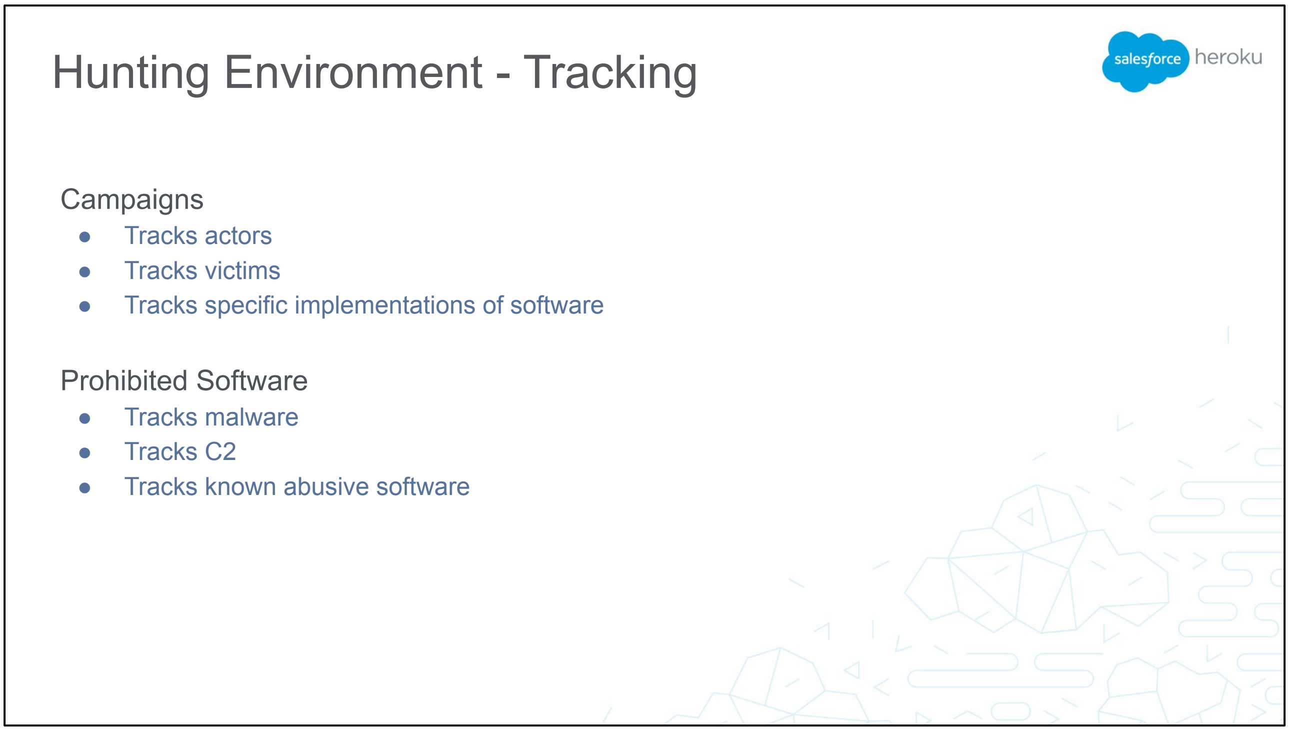
Our malware analysis tools look for footprints in the apps themselves, in either the static state or running state. It also has the capability to kick off other processes, and send logs.

Ticketing System

We have our own ticket system. Standard stuff so that we can queue work and gather abuse reports. There is a lot of toil here, and it is not immune to our “reduce toil” requirement, but changes here take a long time. They do get made though, and that is important.

Suspension Tools

Finally we have suspension tools, this is the tool that has the ability to suspend users and apps. It can also kick off other processes, and send logs. More on this in the next section.



Hunting Environment - Tracking

Campaigns

- Tracks actors
- Tracks victims
- Tracks specific implementations of software

Prohibited Software

- Tracks malware
- Tracks C2
- Tracks known abusive software

We track two general types of activities, campaigns and prohibited software.

Campaigns track actors and/or victims, and custom bespoke software that they have written.

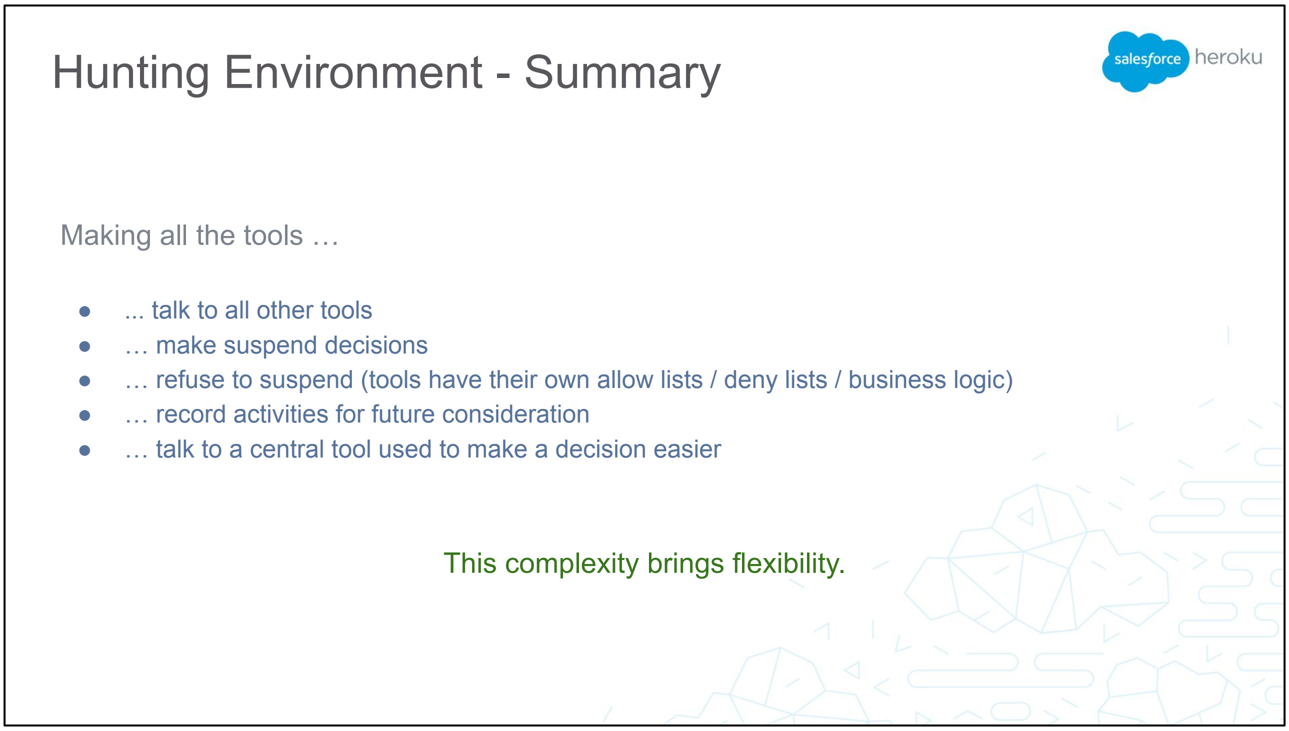
Prohibited software looks for known malware, tracks C2 communications and similar artifacts, and known abusive software that anyone can install, including those with a "Deploy to Heroku" button.

********************* extra info **************************

Campaigns are usually about tracking actors. For example, phishers that keep returning with more than one phishing kit. Sometimes it is easier to track victims, for example, credential stuffing attacks against another site, tracked under the name of the target site rather than trying to separate out the actors. We also end up tracking specific implementations of software here when the software has not been commoditized, meaning some of the quick and dirty code written by the actors for a particular malicious purpose.

Prohibited software is tracked slightly separately, and usually includes software that anyone can install. Cryptomining software, torrenting software, and similar are tracked here. A lot of the badness that is installed via a "deploy to Heroku" button ends up getting tracked and actioned here. Remember, just because it has a "Deploy to Heroku" button, does not mean that it is sanctioned by Heroku.

There is overlap between these two categories, and even internally we blur the lines. An actor installing prohibited software multiple times is still worth tracking as a campaign so that we can see what they do next. This is also how we keep abreast of new developments before they become a problem.



Hunting Environment - Summary

Making all the tools …

- ... talk to all other tools
- … make suspend decisions
- … refuse to suspend (tools have their own allow lists / deny lists / business logic)
- … record activities for future consideration
- … talk to a central tool used to make a decision easier

This complexity brings flexibility.

Let’s put it together.

Any tool can “talk” to any other tool, even if it is the machine equivalent of, “I see bad. Do you see bad?”

Any tool can tell the suspension tools that any user or app needs to be suspended.

Suspension tools have their own allow and deny lists and internal business logic to help identify and stop mistakes before they become costly.

And everything that these tools do is recorded.

This may be complex, but this is what gets us the flexibility we need without taxing any singular system.

******************** extra info *************************

Suspension business logic can be really simple, like lists of non-allowed domain names. Or really complex ones, implemented as business logic. Like, a VIP account that has long history and is in good standing should get a support ticket instead of being suspended outright.

Every action any tool takes should be recorded for all the usual measurements and metrics reasons, but also for later consideration with new tools.

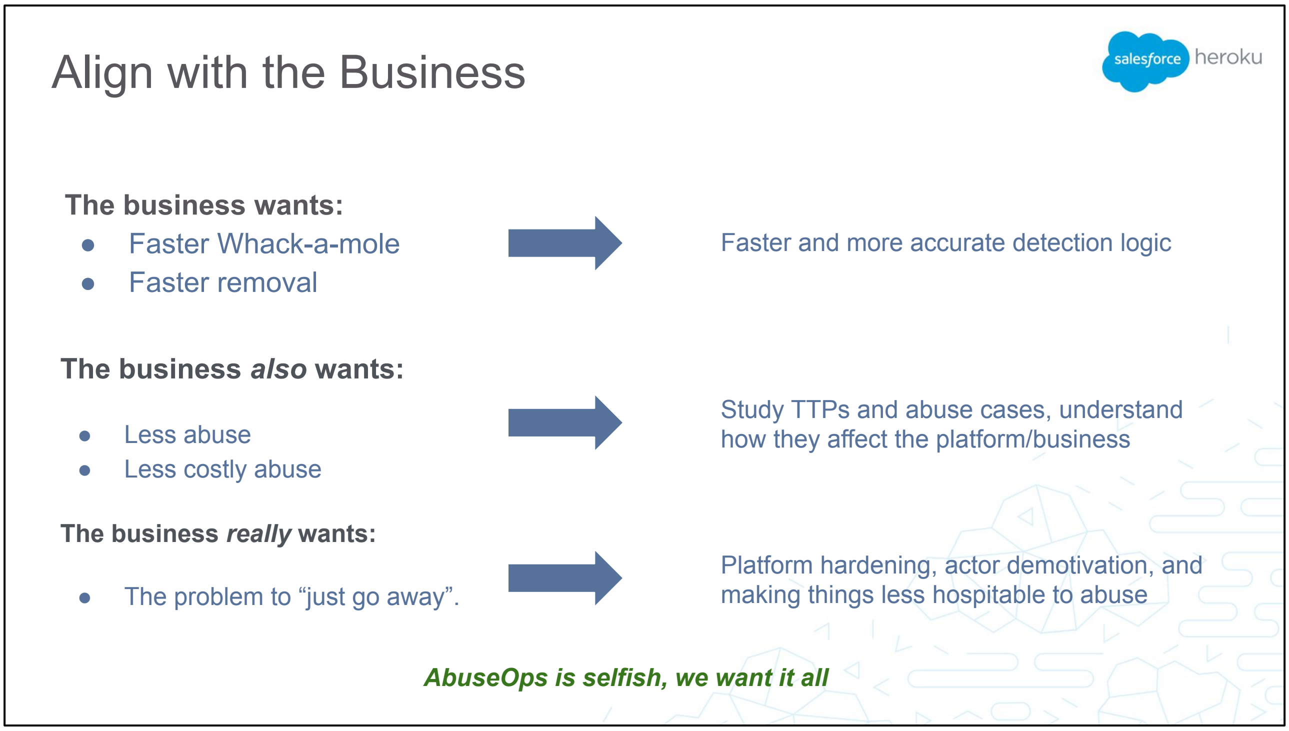
When a footprint spans multiple tools, even though they may be triggering each other, it is often easier to create a central authority to implement a “meta rule” to make the final decision. This is very similar to a classic SIEM model. However we are looking to improve this path with a richer communications system to avoid putting even more burden on our SIEM infrastructure. THINK distributed SIEM, and SIEM as originally defined, not as some singular product.

Previous decisions help drive future decisions. This is part of the feedback loop. A simple example is that if every usage of a particular IP address has belonged to a bad actor, then every future usage should be examined automatically. Further, it can be raised as a new indicator for inclusion into new footprints.

Flexibility is the key. The tools should always be questioned, measured and reported on. New data sources are implemented all the time, and as in any high performance environment, changes happen that can render your hunt obsolete, or worse; completely broken. Stay on top of it.

To do this correctly, there will be “care and feeding” requirements.

Responding and Reacting



Align with the Business

The business wants:

- Faster Whack-a-mole
- Faster removal

→ Faster and more accurate detection logic

The business *also* wants:

- Less abuse
- Less costly abuse

→ Study TTPs and abuse cases, understand how they affect the platform/business

The business *really* wants:

- The problem to “just go away”.

→ Platform hardening, actor demotivation, and making things less hospitable to abuse

AbuseOps is selfish, we want it all

We align with the business. They of course set the tasks and the priority, but it is up to us to be able to interpret them.

The business wants us to remove abuse faster. Fair Enough. This means we need faster and more accurate detection logic.

The business also wants less abuse, or at least less costly abuse. That requires us to study the actors, what they are doing and how they affect the platform and the business.

What the business *really* wants is the problem to “just go away”. This seems simplistic, but after giving it some thought, we realised that there is a path here, via platform hardening and demotivating actors.

If we make the entire thing less hospitable to abuse, maybe we can get close.

We in AbuseOps are selfish. We want all of this.

********************* extra info **************************

We joke about “what the business wants” but really it is about different aspects of the business requiring different things and prioritizing different things. Each department, team or organization may have a differing view. For this talk, we’ll still call it “The

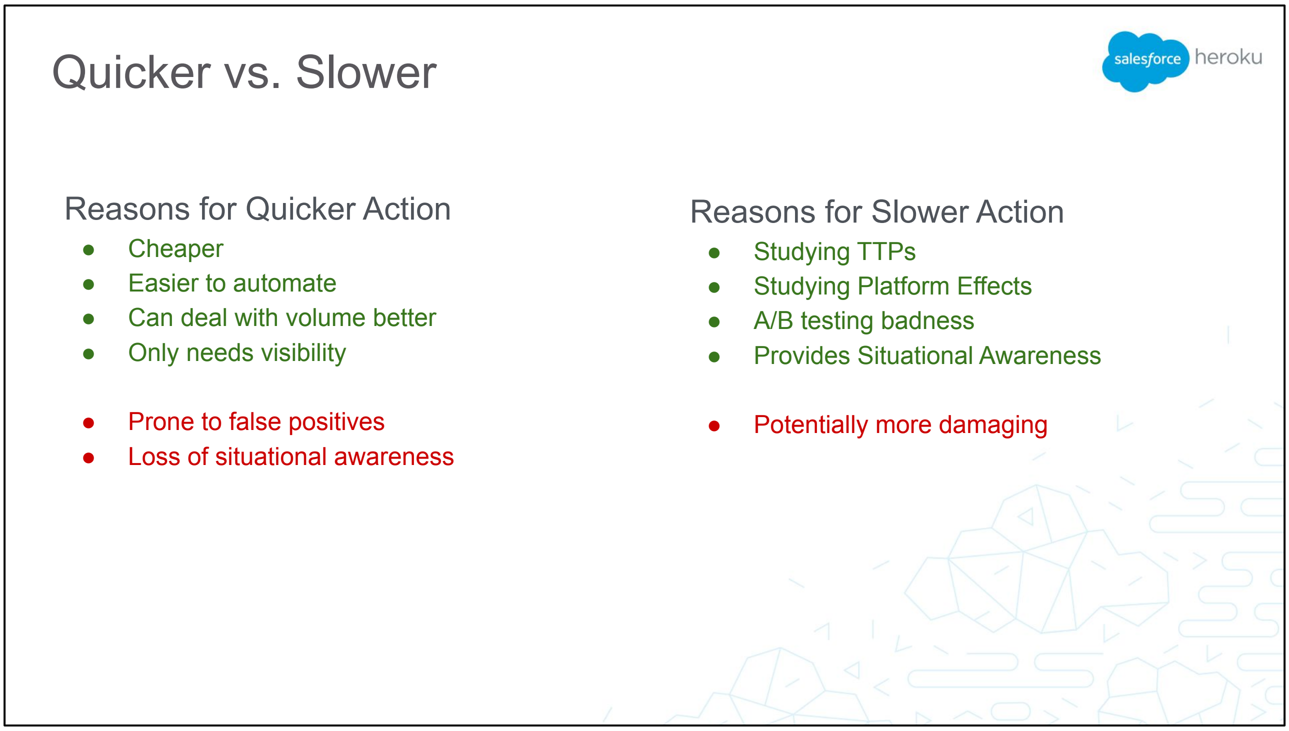
Business” but understand there is complexity there.

The business expects a certain cadence to removing abuse. The earliest sentiment, especially in the early days of abuse operations is to “do the needful” faster. Faster Whack-a-mole, faster removal, faster firefighting. This is a tricky time, because if you get used to doing nothing but firefighting, firefighting is all that you will end up doing. So the real goal here should be the building and improvement of the detection logic and tools, so that you can turn around investigations faster, move towards automation, and speed up that loop. “Never see the same abuse twice”, right?

The business also wants less abuse and less cost to abuse. There does seem to be two measures of cost here, unrealised billings and cost to serve. In other words the amount of money failing to come in, and the amount of money being paid out, because of abuse. To be able to tackle this aspect of the problem requires some study. Studying the tactics, techniques and procedures and understanding how they affect the platform and the business is needed. The results can be fed into loop loop, and and can be used as the start of some good metrics.

There is a fun little point that *someone* in the business will reach, which is that moment of ... *heavy sigh* followed by, “why won’t they just go away”. It’s Reporting Fatigue. When you report the same thing happening over and over again, they get annoyed / bored / frustrated as well. However, at this point it becomes possible to explore what it will take to “make it go away”. And this usually entails platform hardening, devaluation of loot, actor demotivation, and making things less hospitable to abuse. This is hard.

Finally, we in AbuseOps are selfish. We want all of this. And More.



Quicker vs. Slower

Reasons for Quicker Action

- Cheaper
- Easier to automate
- Can deal with volume better
- Only needs visibility

- Prone to false positives
- Loss of situational awareness

Reasons for Slower Action

- Studying TTPs
- Studying Platform Effects
- A/B testing badness
- Provides Situational Awareness

- Potentially more damaging

The first two items on the list in the previous slide have competing requirements.

The first requires us to respond quicker, and the second to study the actor and their actions and hence respond slower.

There are pros and cons to each strategy, the big ones being that quicker is usually easier but prone to false positives, while slower provides better situational awareness, but since the actor may still be active it could be more damaging.

We have found that is sometimes worth switching back and forth depending on what we see, and we have also found that going slow first does allow us to act quicker later, with accuracy.

******************** extra info *************************

The first two items in “the business list“ have competing requirements. The first requires us to respond quicker, while the second requires us to study the actor, and hence respond slower.

There are pros and cons to each strategy. Responding quicker will result in keeping the cost of abuse down, it is easier to automate and because of that, it is easier to deal with volume.

However, it is more prone to false positives and can result in the loss of situational awareness since the results of fully automated systems are usually logged in some report at best, and are rarely looked at.

On the other hand, if you slow down and study what the actors are doing, and how they do it, you have a chance at really making an effect by choosing a course of action that will be devastating to the actor’s activities. Also, you have a chance at better understanding what is happening to your systems, and hardening against the abuse altogether. The con is that the longer the actor is on the platform, the more potential damage they can do and the more it may cost the business.

Again, abuse operations is greedy. We want it all.

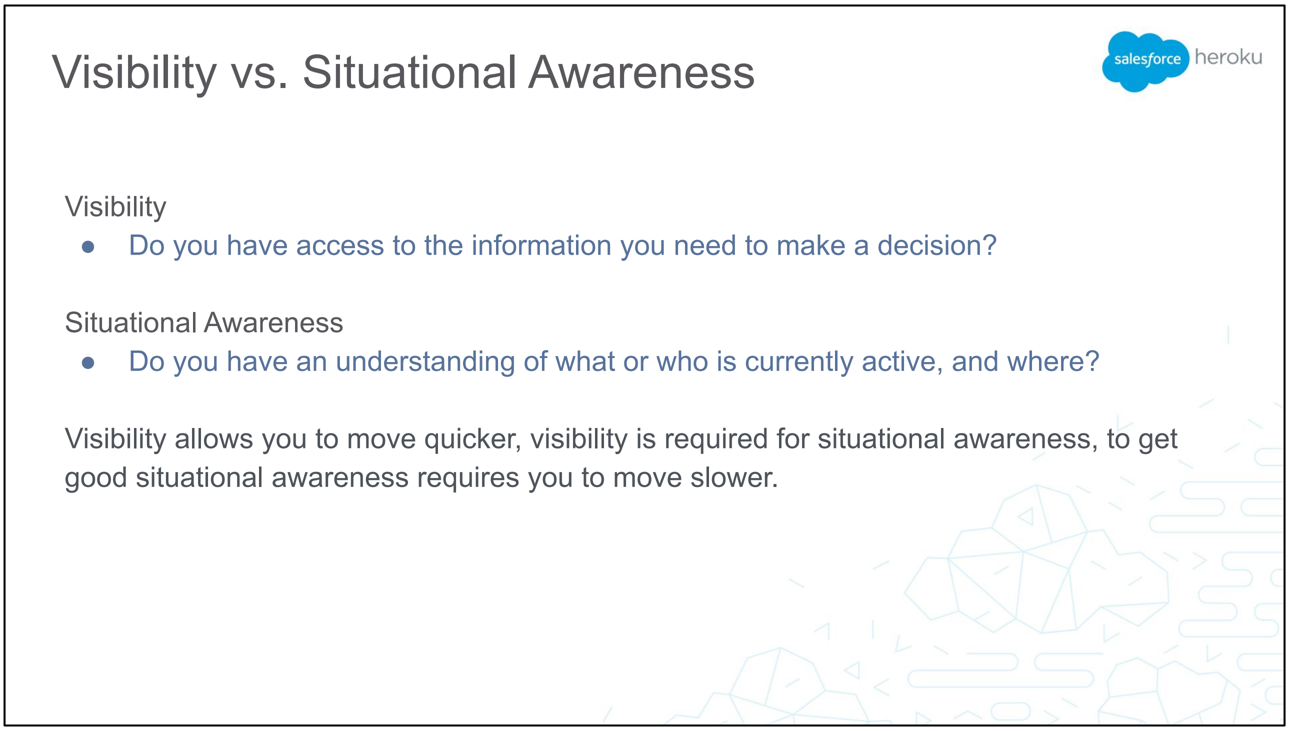
We will inherently choose a strategy that make sense to the type of activity we are seeing and how often we are seeing it. Sometime we write up a detection footprint and automatically action the results in minutes, sometimes we study the actions and the effects for a while to figure out what they are doing and how.

To mitigate the false positive rate of quick actions, we usually deploy new detections in an “alert only” mode until we are satisfied that there will be no false positives.

To limit the damage when we move slow, especially with “actor badness” that requires multiple accounts, we may suspend some subset of the accounts, and study the rest. There is a bit of a bonus here, in that we have noticed that there is potential for a sort of A/B testing. If we suspend half of the accounts on an obscure piece of criteria, will that actor notice? Can this be used to change their behaviour? The answer is “Yes”, in a crude sort of way.

There are more things we do here as well, but we are skirting dangerously close to our secret sauce. Besides, if you are exploring these kinds of things right now, we would love to have a conversation, compare notes, and see if we have come to the same conclusions.

On a final note, we have found value in switching between the two strategies, depending on how the actors change their behaviour. Actor TTPs are not static, your activities and responses should not be either.



Visibility vs. Situational Awareness

salesforce heroku

Visibility
- Do you have access to the information you need to make a decision?

Situational Awareness
- Do you have an understanding of what or who is currently active, and where?

Visibility allows you to move quicker, visibility is required for situational awareness, to get good situational awareness requires you to move slower.

Moving quicker requires visibility. Situational awareness not only require visibility, but also requires studying the actors, and hence going slower.

So overall, we end up going slower to ultimately go faster, with accuracy.

But when we first started out, we had to go fast just to get our heads above water.

Understand that there is some hidden complexity here.

********************** extra info ***************************

It is noteworthy that “Moving Quicker”, requires visibility. Situational awareness also requires visibility, and the study steps of “Moving Slower” is what gets you situational awareness.

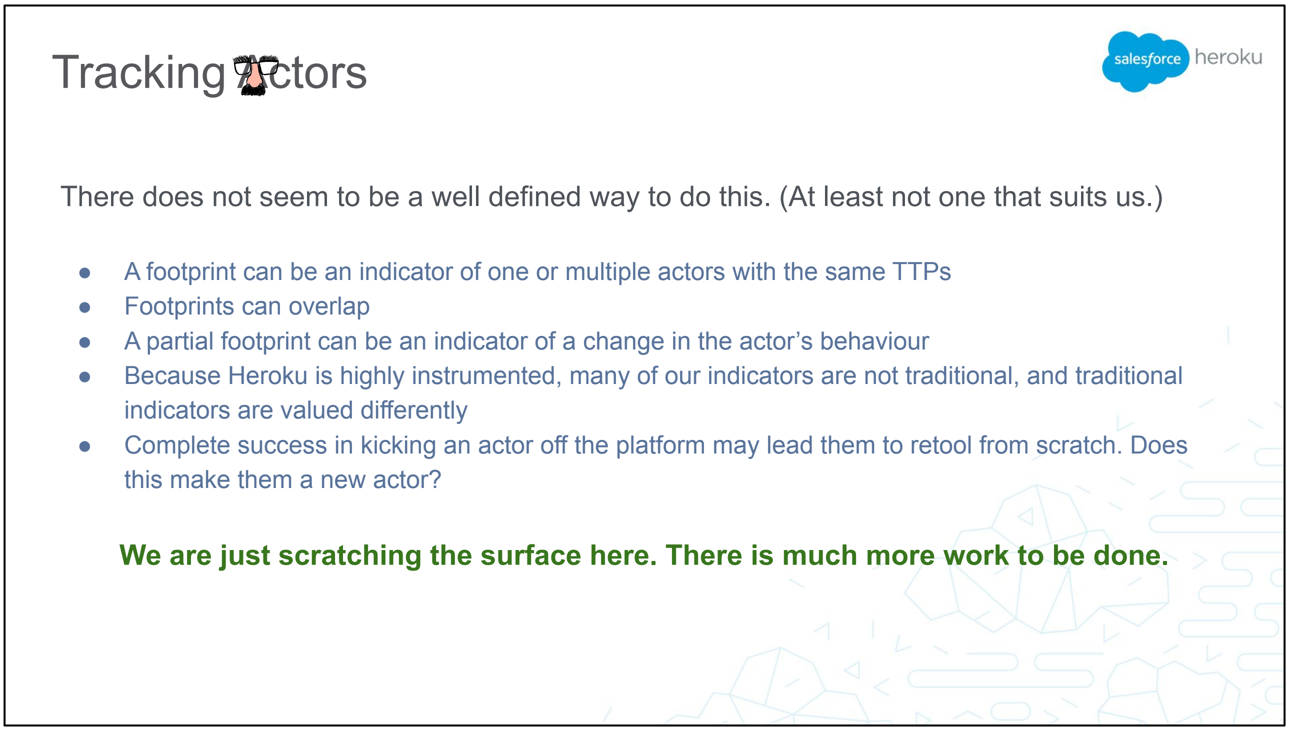
Situational awareness is also partially derived from the logs of previous actions, and we have found that the volume of logs from quicker actions are going to be bigger than those of slower actions. Naturally.

Add in the fact that quicker actions are more prone to false positives, and one quickly realises that the situational awareness you may have derived is actually incorrect.

The full relationship between the two is more complex than initially expected, and we have not explored all the outcomes and ramifications of this beyond the head of the singular analyst.

It seems that an analyst that is steeped in the system intuits a lot of this, and the only thing we are missing is the ability to articulate it. Maybe.

We will have to look into this deeper because...



Tracking Actors

salesforce heroku

There does not seem to be a well defined way to do this. (At least not one that suits us.)

- A footprint can be an indicator of one or multiple actors with the same TTPs
- Footprints can overlap
- A partial footprint can be an indicator of a change in the actor's behaviour
- Because Heroku is highly instrumented, many of our indicators are not traditional, and traditional indicators are valued differently
- Complete success in kicking an actor off the platform may lead them to retool from scratch. Does this make them a new actor?

We are just scratching the surface here. There is much more work to be done.

Tracking actors is hard. Tracking persistent returning actors is haaarrrrd.

Ok, not quite, but there does **NOT** seem to be a well defined way to track actors by footprint as we defined it that gives us the flexibility we desire.

We want to know if an actor changes behavior and we want to know in an automatic way.

- And when they do, we want to be able to add a new indicator to the footprint in as few steps as possible.
- We want to be able to split and merge footprints as we track multiple actors.
- We want this to be a tool just like the rest, remember the requirements back in the tools slide?

We want a lot.

Today, we track mostly manually, and this is a lot of toil.

We are looking to build a system that enables us to spot returning actors that have changed a TTP (AKA a pivot),
or spot when a footprint is covering for multiple actors (AKA a splash)
or …. Fingers crossed … spot an actor that returns and starts from scratch (AKA retools).

********************* extra info **************************

Tracking actors is about situational awareness.

We want to know if an actor changes behaviour, and we want to know in an automatic way. And when they do we want to be able to add the new indicator in as few steps as possible to the footprint. We also need to be able to age out or otherwise manage indicators that no longer apply, and be able to map individual indicators to multiple footprints, and that in turn to one or multiple actors.

In the extreme case, when we successfully evict a persistent actor, and they completely retool and return, they will get a completely new footprint, and may be tracked as a completely new actor. So any system used to track actors will also need to be able to merge actor “profiles”, and have multiple footprints per actor.

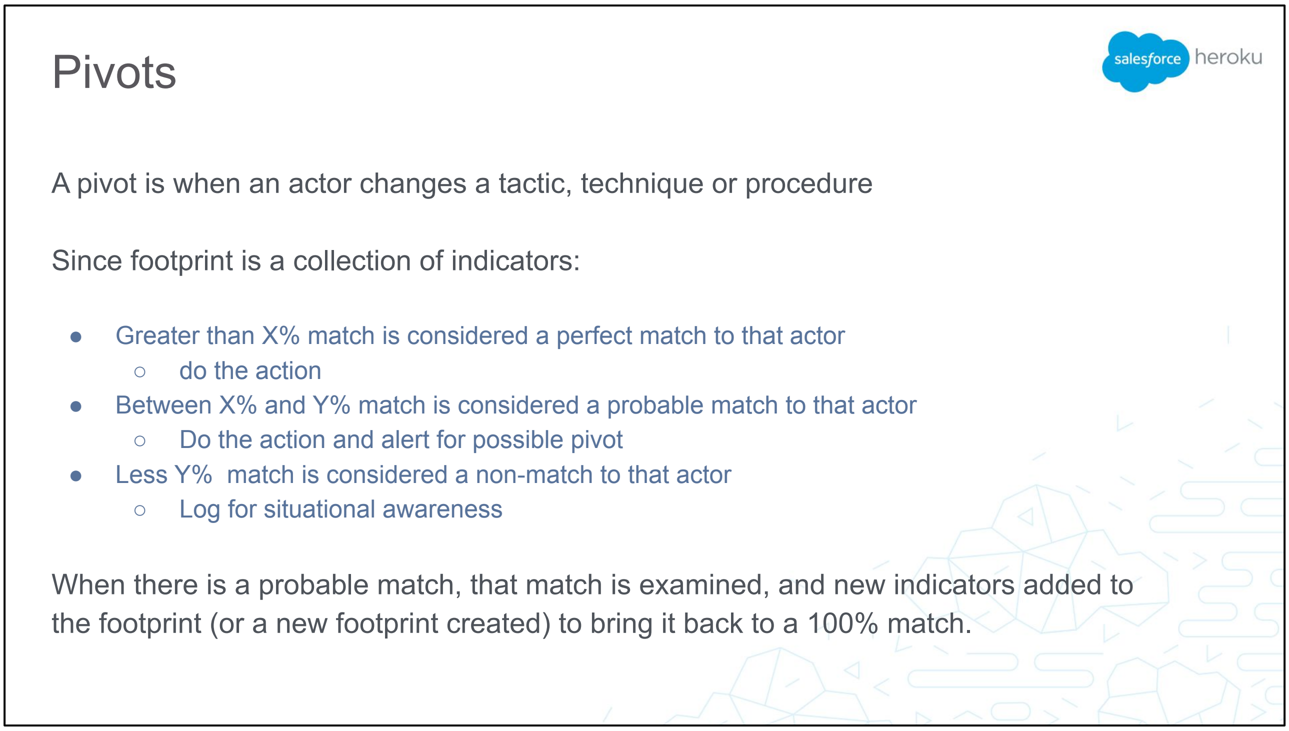
But if that is the case, we will also need to track historic footprints.

Today we do all of this with a combination of manual work, and making things inherent to how the platform functions. And this is not good enough.

If we can make this functional, not only will we gain and continue to have situational awareness, but we can also start to automate our automation! The DREAM.

This is an area we will be working on in the coming year.

Let’s dive into a couple of the more interesting things that will make tracking actors not only interesting, but fun.



Pivots

A pivot is when an actor changes a tactic, technique or procedure

Since footprint is a collection of indicators:

- Greater than X% match is considered a perfect match to that actor
  - do the action
- Between X% and Y% match is considered a probable match to that actor
  - Do the action and alert for possible pivot
- Less Y% match is considered a non-match to that actor
  - Log for situational awareness

When there is a probable match, that match is examined, and new indicators added to the footprint (or a new footprint created) to bring it back to a 100% match.

A pivot is what we call a change to a TTP of a returning actor (different from a pentester pivot). If a footprint is a collection of indicators, then given enough of them, a partial match usually indicates a change in behaviour by the same actor.

Actors are people too. When they are suspended, they will test how, by isolating for variables and making small changes one at a time until they "get it working again".

So a form of threshold detection, like shown, works really well as an accurate way to detect their return and take the same action.

Then updating the footprint with new indicators reflecting the change they just made keeps the footprint accurate. This is designed to make it difficult for a persistent actor to regain a foothold on the platform.

And frustrates the heck out of the actor, especially if we cut them off
- at each pivot
- before they profit.

******************** extra info *************************

A pivot is what we call a change to a tactic, technique or procedure (different from a pentester pivot). If a footprint is a collection of indicators, then given enough of them, a partial match usually indicates a change.

Examination of a partial match and discovery of what has changed will reveal more indicators, and the addition and redefinition of the footprint with the new indicators will bring one back to a 100% detection rate.

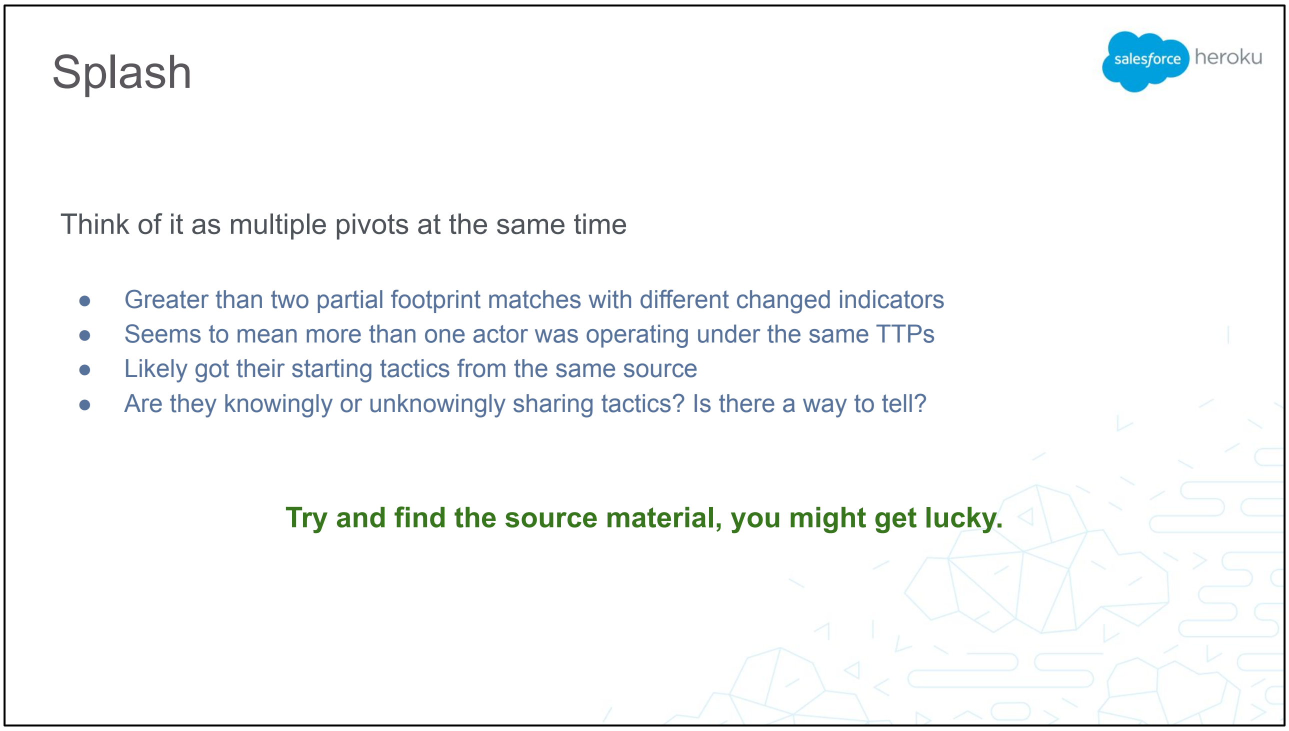
The reason that this works for most actors is that they are developers like any other. When presented with a problem, like their account getting suspended, they will isolate for variables, redo their work changing one thing at a time, and test what they believe may have been the thing that got them caught.

Tied together with automation, and an appropriate level or automatic suspension, the system can send alerts for “close enough” suspensions for review.

There can also be multiple watermarks for how much of a match there needs to be to do each of the things, but having two levels seems to work quite well.

- Greater than X% of indicators, suspend automatically.
- Between X% and Y%, alert as a partial match.
- Under Y%, log and ignore.

How to choose the levels still seems a bit of an art. However, two things to consider when doing so, are the number of indicators in a footprint (higher numbers seem to allow a lower threshold for partial matches without false positives), and having different levels per actor, for at least the long running actors that may have been hiding for a while. They have a tendency to understand the platform a bit better than most.



Splash

Think of it as multiple pivots at the same time

- Greater than two partial footprint matches with different changed indicators
- Seems to mean more than one actor was operating under the same TTPs
- Likely got their starting tactics from the same source
- Are they knowingly or unknowingly sharing tactics? Is there a way to tell?

Try and find the source material, you might get lucky.

Sometimes instead of seeing a singular pivot, we see multiple pivots but with different partial footprints.

We call this a splash. Ever slap a mud puddle?

This seems to mean that there are multiple actors operating under the same set of TTPs.

We have found great value in looking for how they are sharing these tactics. Are they all reading the same forums? Or, all using the same instructional youtube video?

When we find that source then we can have some more fun.

Imagine knowing what they are going to do before they finish implementing it (especially a code change), and adding those as indicators to the footprint before the changes are deployed.

Frustrating them as well.

Shout-out to our fantastic Threat Intel team. They help find the stuff out there.

********************* extra info **************************

Sometimes, after suspension, instead of a one-change pivot we will suddenly see new accounts with multiple partial footprint matches all at the same time, but not the same subset of indicators.

Some analysis of these have shown that there was actually more than one actor under that original footprint, and since they were all caught out and kicked from the platform (usually twice), they are looking for a way back and are again trying to figure out how they got caught.

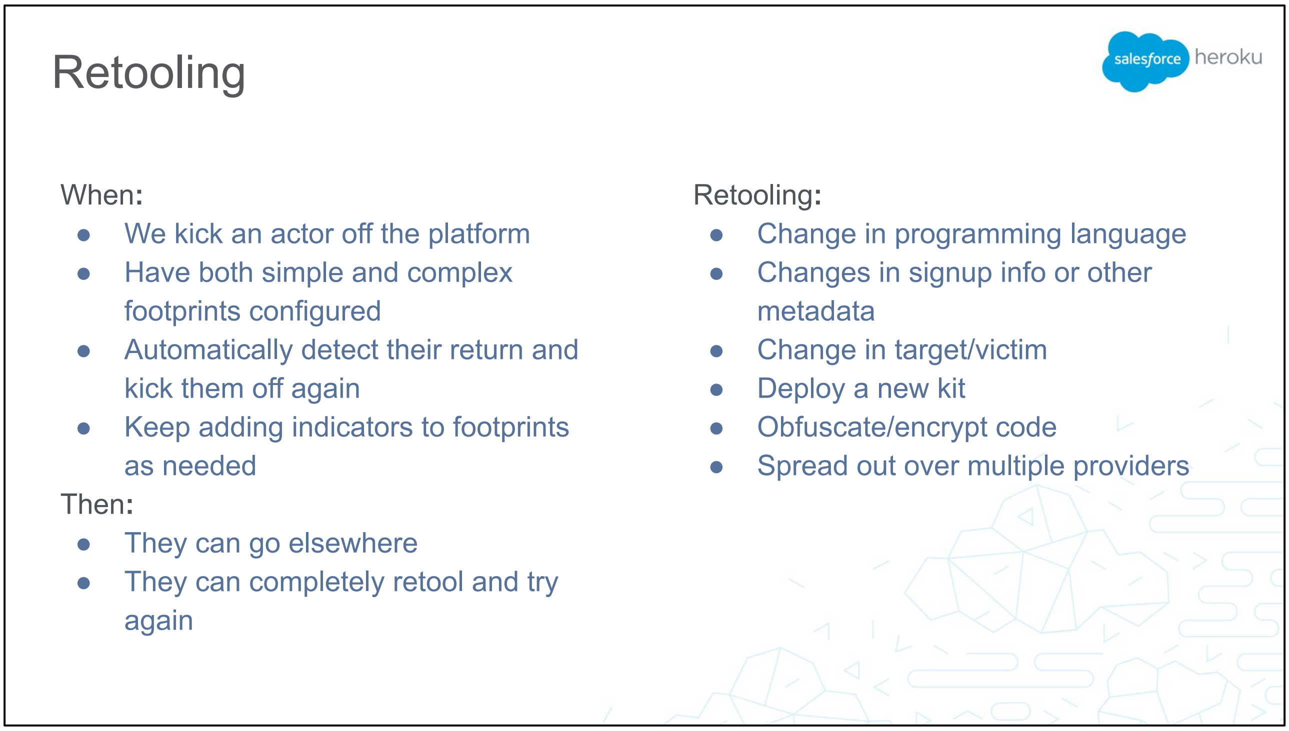
How did they start with the same TTPs? Sometimes they simply started with the same how-to document someone published, sometimes they all got the same software or the same kit.

Actors that converge to a new set of working TTPs, and get back on the platform, do appear to be more likely to be sharing information in some way, whereas actors that bounce around for a while don't, especially when you can see that they are each making the same mistakes.

One interesting scenario that comes up often enough is where multiple actors are sharing TTPs via some forum, and *racing* each other to "find a fix" to the suspension problem. They may each be looking for the glory of fixing the problem, and hence not sharing. At least not until they have found it.

If you can find the forum, or the how-to, or whatever, and watch the fun, you can easily stay ahead of them by updating your footprints and detection logic before they finish implementing a new TTP. And the bonus here is that you can do it with little effort. Less effort than it takes the actor to change their TTP.

Do it right, and you can cause a lot of "banging of chest" and "gnashing of teeth".



Retooling

When:
- We kick an actor off the platform
- Have both simple and complex footprints configured
- Automatically detect their return and kick them off again
- Keep adding indicators to footprints as needed

Then:
- They can go elsewhere
- They can completely retool and try again

Retooling:
- Change in programming language
- Changes in signup info or other metadata
- Change in target/victim
- Deploy a new kit
- Obfuscate/encrypt code
- Spread out over multiple providers

- When we have all the tooling in place,
- when pivot detection is working as designed, and
- When we manage to stop actors before they have even begun,

then they have only two choices: Go somewhere else, or retool.

When they retool they change significant portions of their

- apps,
- code,
- infrastructure, and
- behaviours

that make them indistinguishable from a new actor. This used to bother us quite a bit, but then it occurred to us, “So what?”

The only reason that we could think of that we needed to tie two actors together was for law enforcement purposes, so that we could collect more evidence. But this does not apply to all actors, and is kind of rare.

So, retooled actors end up being identified as new actors just by the nature of how our hunt works.

And because we are pretty thorough in our indicator collection, sometimes we still find evidence that a new actor is in fact a returning actor that has retooled.

next…

Then the only avenue they have is to go elsewhere, or retool.

Retooling usually means starting from scratch, and rebuilding whatever infrastructure they were running in some sort of novel way. We have seen changes in language, changes in how they sign up to the platform, changes in targets, attempts at obfuscation or encryption of code, deployments of new kits, or spreading their kit across multiple services providers.

It can be really difficult to tell if a new “badness” detected on the platform is truly new, or if it is previous actor that has retooled. And this was bothersome for a while, until we realised that it does not matter all that much. All that really matters is, “Is there a new footprint that needs to be deployed?”

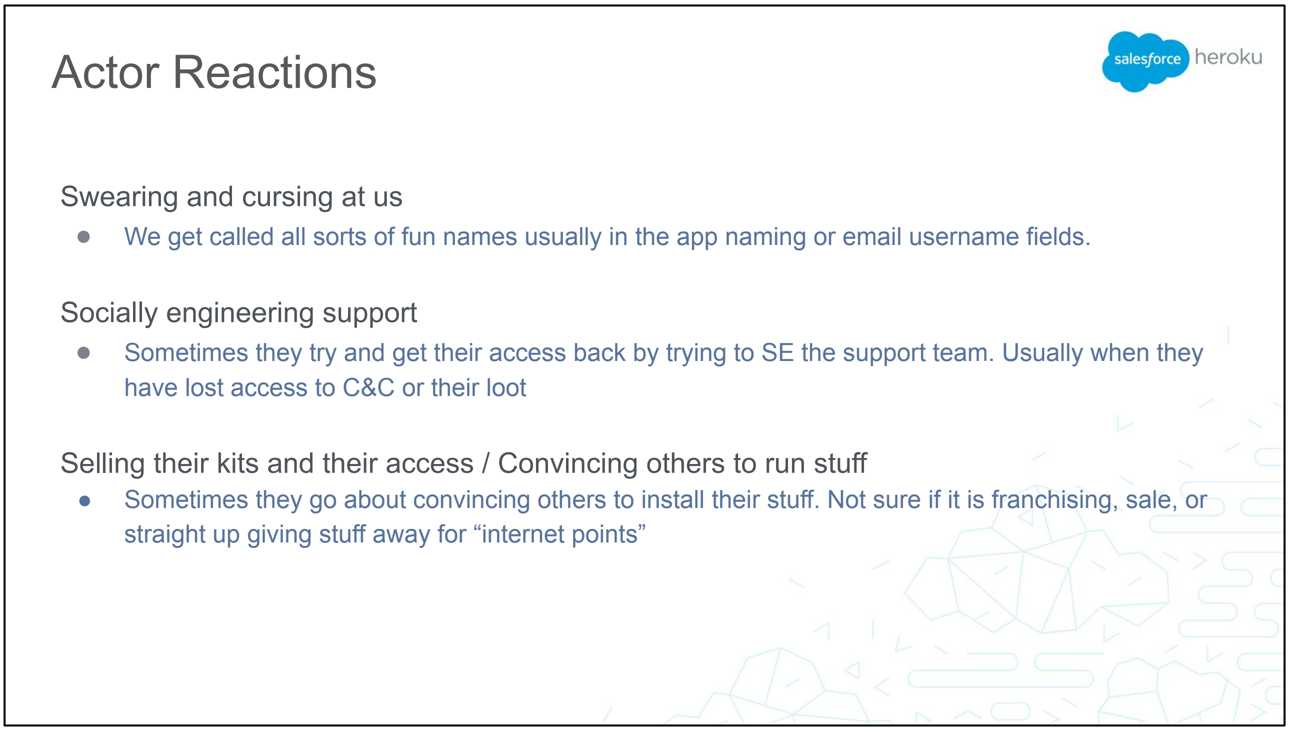
We can go back to our hunt loop, follow the steps, and build a new footprint under a new actor profile, and worry about tying that to old actors later.

Sometimes the very process will lead us to evidence showing that it is indeed the same actor retooled.

Things like:

- A change in programming language, but with the same naming convention and code profile.
- A change in credit card, but still the same credit card provider.
- Obfuscation or encryption of code? This one is odd, because the technique sticks out like a sore thumb, it does the opposite of hiding. The code still has the same communications profile.
- Spreading out over multiple providers? Although this shrinks their footprint on our system, once we have latched onto the communication profile, we can usually tell if it is the same actor or not.

We do ultimately want to know if it is the same actor, because it will help us to devalue loot, and demotivate them, and in the case of pursuing legal action, provide more evidence.



Actor Reactions

salesforce heroku

Swearing and cursing at us

- We get called all sorts of fun names usually in the app naming or email username fields.

Socially engineering support

- Sometimes they try and get their access back by trying to SE the support team. Usually when they have lost access to C&C or their loot

Selling their kits and their access / Convincing others to run stuff

- Sometimes they go about convincing others to install their stuff. Not sure if it is franchising, sale, or straight up giving stuff away for “internet points”

Actor reactions are fun.

Some actors call us names.

Others open tickets and try to socially engineer support to get their loot or infrastructure back.

Others, maybe seeing the writing on the wall, realise they have squeezed out what they can and got about convincing other to run their kits. “Look what I did and you can do it too!” It’s like they have found their own replacement for that actor’s footprint.

********************* extra info **************************

Actor Reactions.

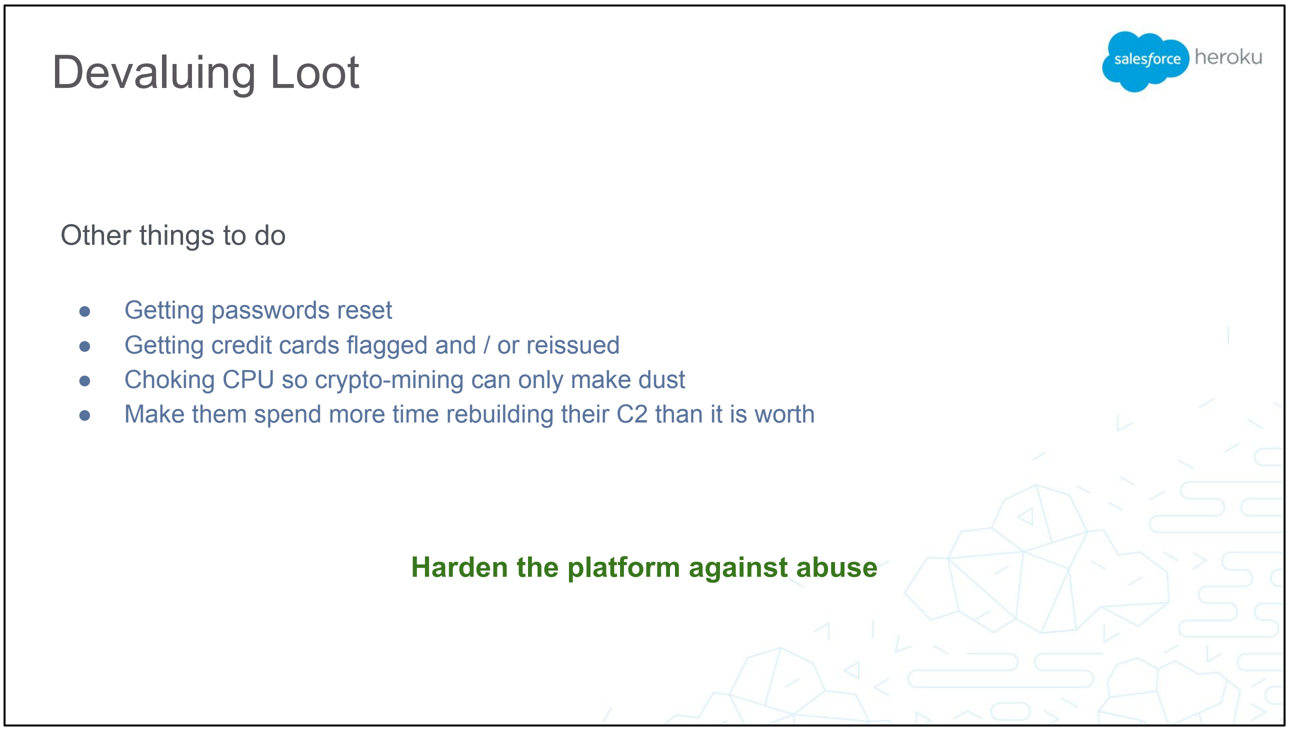
Other than trying again, some of the reactions from actors we have shut down are somewhat interesting.

Sometimes, when they come back and create new accounts or new apps, they will call us all sorts of nasty names. They will also occasionally do it in forums on the Internet. We love this; their tears sustain us. But my personal favourite is when they complain to the creator of the app and blame *them* for getting suspended.

Sometimes they try and convince others to run their stuff. Someone else creates an account, tries to do exactly the same thing as a known actor, and is detected long after that actor has been kicked off the platform and gone dormant. They can usually be spotted because they are sloppier in their opsec, and less persistent than the previous actor.

Sometimes they open a ticket, claiming everything from ignorance to outright threatening us telling us that we must reinstate them or their app. The ones that stick out to me are those that are blatantly and openly phishing and honestly don't think they are doing anything wrong. At least they *seem* honest. They are still not getting their account back.

I need to give a shout-out to our support team, they are an absolutely fantastic crew.



Devaluing Loot

Other things to do

- Getting passwords reset
- Getting credit cards flagged and / or reissued
- Choking CPU so crypto-mining can only make dust
- Make them spend more time rebuilding their C2 than it is worth

Harden the platform against abuse

That last business category, “wanting the problem to go away” can be achieved by devaluing loot.

If we devalue the actor’s loot

- by making it worthless after the fact, like reporting stolen credentials (responsibly),
- or making “it not worth the effort”, like limiting CPU usage so that an actor can’t cryptomine anything of value,

Then it is more likely they WONT return.

We have done this, but so far with mixed success.

This is where hardening the platform come into play. Since we are already studying how the actors act, and how the platform reacts, we recommend fixes to make the platform less hospitable to abuse. We “Push Left”.

Find me later and I’ll tell you a fun story from a previous job about devaluing a phisher’s loot.

******************** extra info *************************

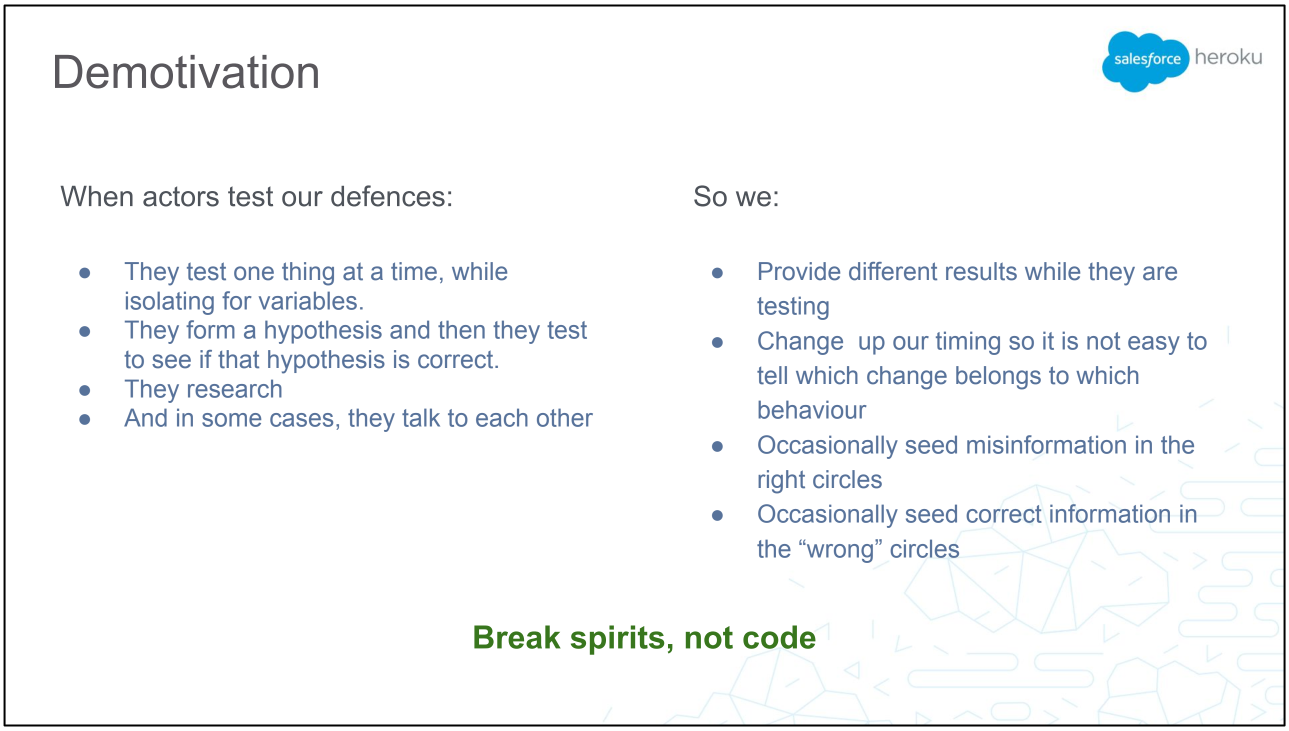
In the category of “what the business *really* wants” : “Wanting the problem to go away”, one strategy is the devaluation of loot.

Loot began as a slang term for stolen credentials. We expanded the term to also mean anything gained through abuse, malice, and crime. This allowed us to include the value of any cryptocurrency, the access to and control of malware (also known as command and control infrastructure), and credit card information to name a few.

One of the things that seems intuitively correct, is that if we can devalue loot effectively, then the actor will be less likely to return. It seems that the trick is to make certain that the actor knows you are doing it, or are willing to do it.

We have done this with mixed success. Find me later, and I’ll tell you a fun story from a previous job.

This also ties into hardening the platform against abuse. There are measures that can and should be put into place that can ruin the ROI for abuse, but as you can imagine, it will be different per abuse category, and some are easier to implement while others involve effort.



Demotivation

When actors test our defences:

- They test one thing at a time, while isolating for variables.
- They form a hypothesis and then they test to see if that hypothesis is correct.
- They research
- And in some cases, they talk to each other

So we:

- Provide different results while they are testing
- Change up our timing so it is not easy to tell which change belongs to which behaviour
- Occasionally seed misinformation in the right circles
- Occasionally seed correct information in the “wrong” circles

Break spirits, not code

On top of devaluation of loot, of killing off ROI, there is an even wider goal of demotivation.

If we really want to keep the actor away, we should work towards making it so that they don’t want to come back.

All of the things we have talked about adds friction to abusive actions ... the monetization of “Free”. And since our ultimate goals are to make abuse less valuable AND less desirable without affecting legit customers, we need to break the spirits of the bad actors.

Break spirits, not code!

<pause>

These two section contained a lot of information. There are more in the notes of these slides. Feel free to download and read them. I hope that there are things in these two sections that resonate with you, and that may help you define what you want or need to do.

But enough of all this philosophy, let’s go back to Spencer with some show and tell!

******************** extra info ************************

A tiny bit of misinformation in the right places can go a long way. It keeps them guessing. Do it right and it can be quite frustrating. We know because some come back and swear at us.

Also in the category of “what the business really wants” : “Wanting the problem to go away” ... we started thinking about what that would take.

One thing became very obvious, very quickly, is pursuing demotivation strategies.

When an actor is testing our defences

- They test one thing at a time, while isolating for variables.
- They form a hypothesis and then they test to see if that hypothesis is correct.
- They research
- And in some cases, they talk to each other

So we

- Provide different results while they are testing, poisoning their results
- Change up our timing so it is not easy to tell which change belongs to which behaviour
- Occasionally seed misinformation in the right places.
- Occasionally seed correct information like “OMGWTF, every time I try to run this code on Heroku, I get suspended!”
- And other things….

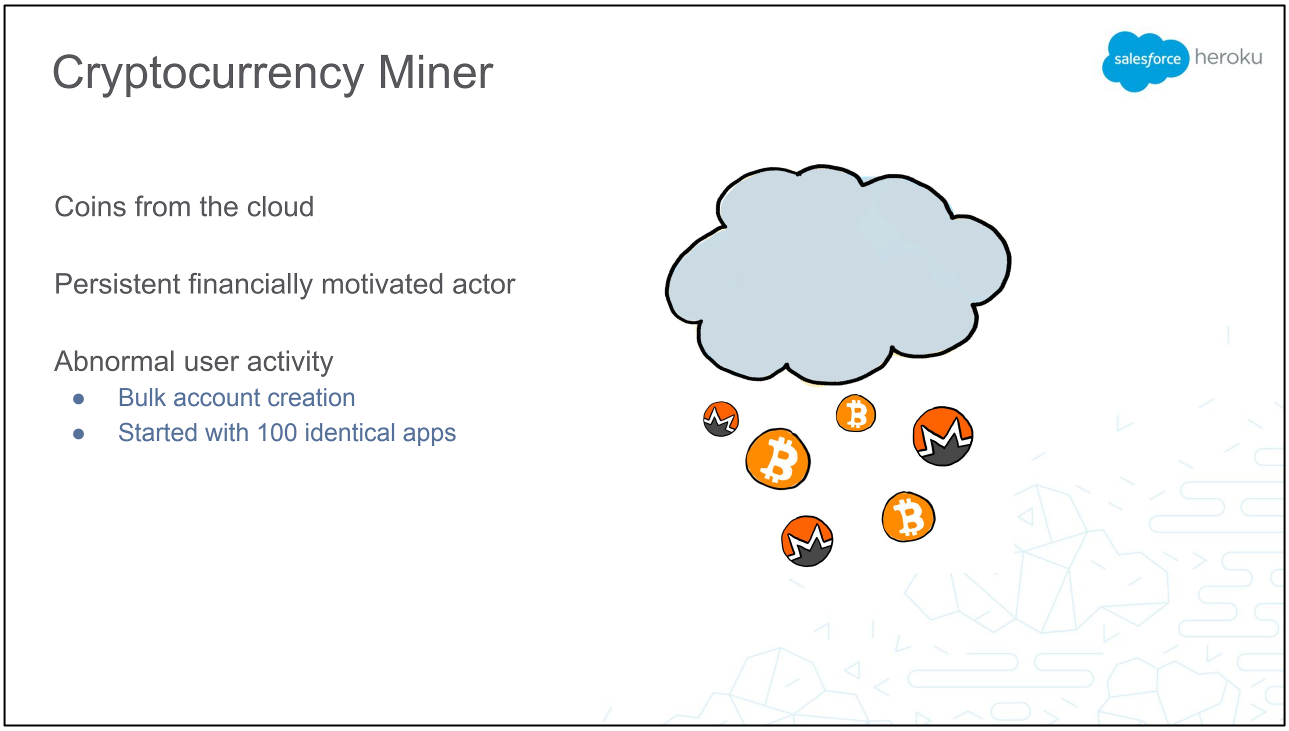
Demotivation is on top of devaluation of loot and killing ROI. Obviously if the only motivation is the ROI, then reducing that will already be demotivating.

This take a lot of time, study and resources to pull off successfully, so we reserve this strategy for only the most persistent of actors. It is difficult to do this and still claim that we are reducing toil. But as we get better, through relentless incrementalism, we should be able to even reduce that.

Break Spirits Not Code!

Enough of all this philosophy, let’s go back to Spencer for a little show and tell.

SHOW & TELL



Cryptocurrency Miner

Coins from the cloud

Persistent financially motivated actor

Abnormal user activity

- Bulk account creation
- Started with 100 identical apps

The first actor is a huge fan of decentralized currency, loves blockchains and hates electric bills.

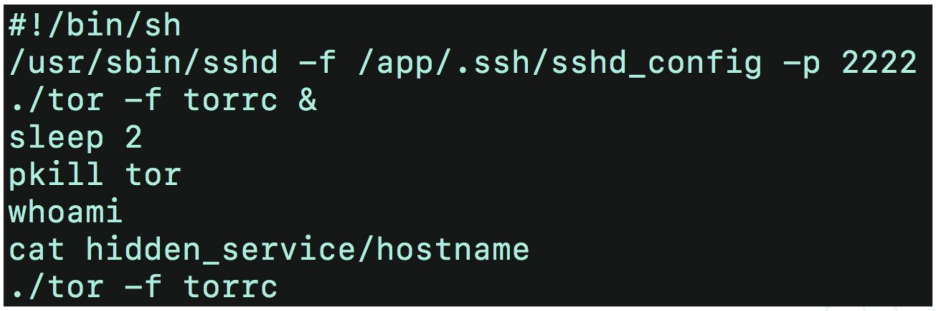
They appeared on our radar by spinning up 100 apps per user account across multiple accounts from free email providers.
(A user account is limited to running five apps until they verify their account by adding a payment method, after that, 100 apps is the max.)

So... new accounts immediately adding a payment method and spinning up 100 apps...
THAT'S NOT NORMAL. (internal indicators!)
This is really not exposing any detection logic. That activity stands out and gets on our radar fast.

And to top things off, fraudulent credit cards were used to verify each account AND all of the apps were identical.

Cryptocurrency Miner

salesforce heroku

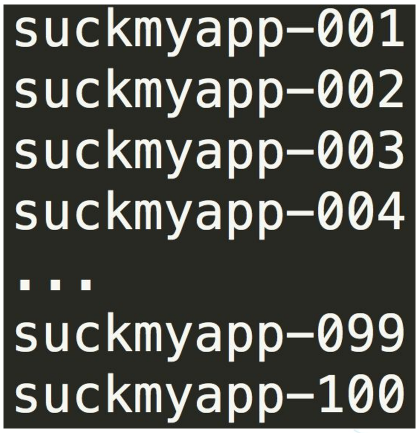


```
#!/bin/sh
/usr/sbin/sshd -f /app/.ssh/sshd_config -p 2222
./tor -f torrc &
sleep 2
pkill tor
whoami
cat hidden_service/hostname
./tor -f torrc
```

So we inspected one of the apps and saw that they were running a Tor hidden service accessible through ssh.

Here’s a screenshot of the script that would run once their app was deployed.

Cryptocurrency Miner



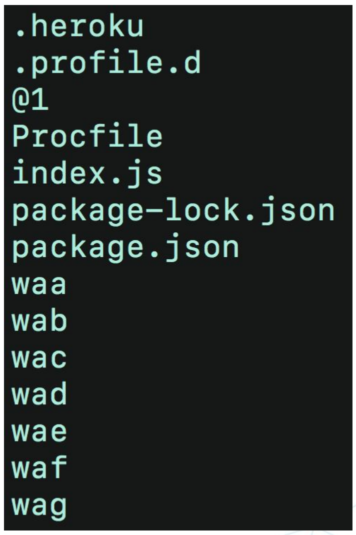
```
suckmyapp-001
suckmyapp-002
suckmyapp-003
suckmyapp-004
...
suckmyapp-099
suckmyapp-100
```

We suspended the accounts and very soon, the actor was back and the cat and mouse began.

One clear sign that “it’s on” with a bad actor is when they start getting cute with their app names.

Obviously, this is the G-Rated end of the spectrum and didn’t really hurt our feelings.

Cryptocurrency Miner

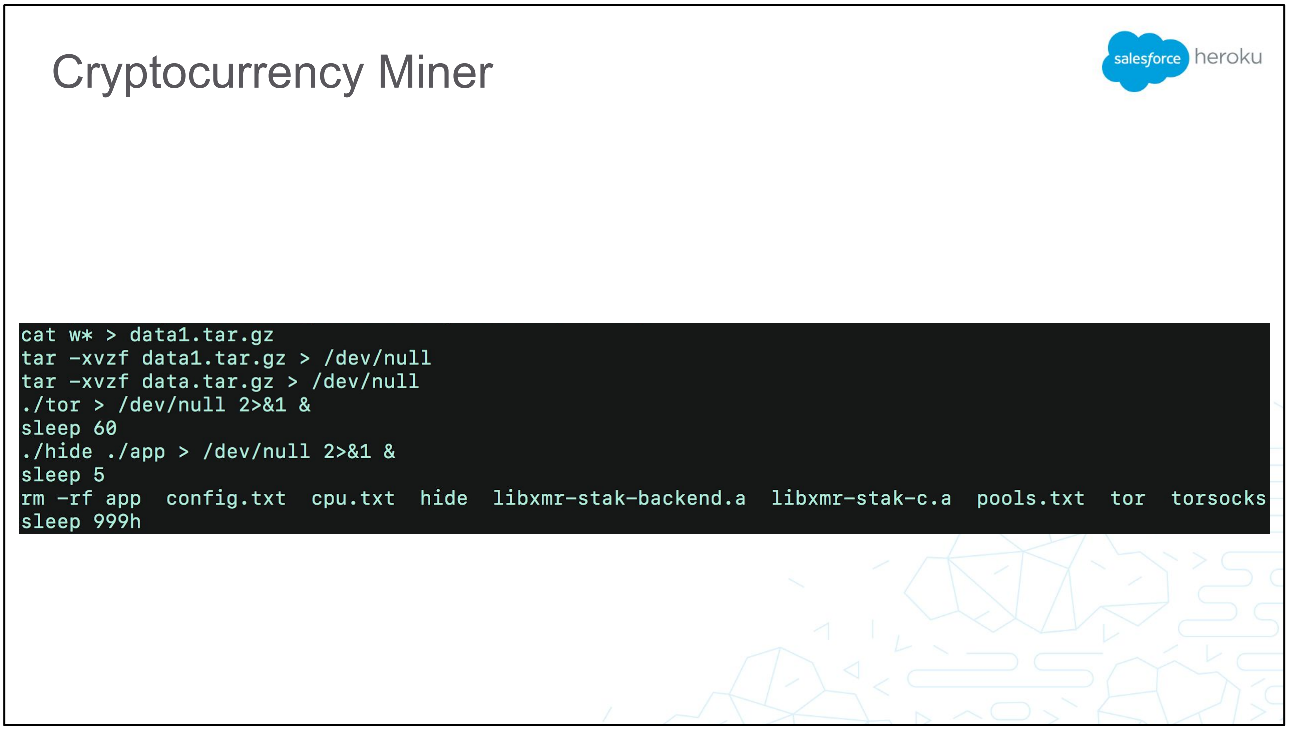


```
.heroku
.profile.d
@1
Procfile
index.js
package-lock.json
package.json
waa
wab
wac
wad
wae
waf
wag
```

After a short while the code changed and we saw what they were really after - mining Monero.

The apps were mining with libxmr and sending all traffic through Tor.
To maximize profit, each app was pegging CPU resources. Fantastic.

After seeing this, we believe that the initial wave of apps was part of some capabilities testing. Some real live bad actor devops.



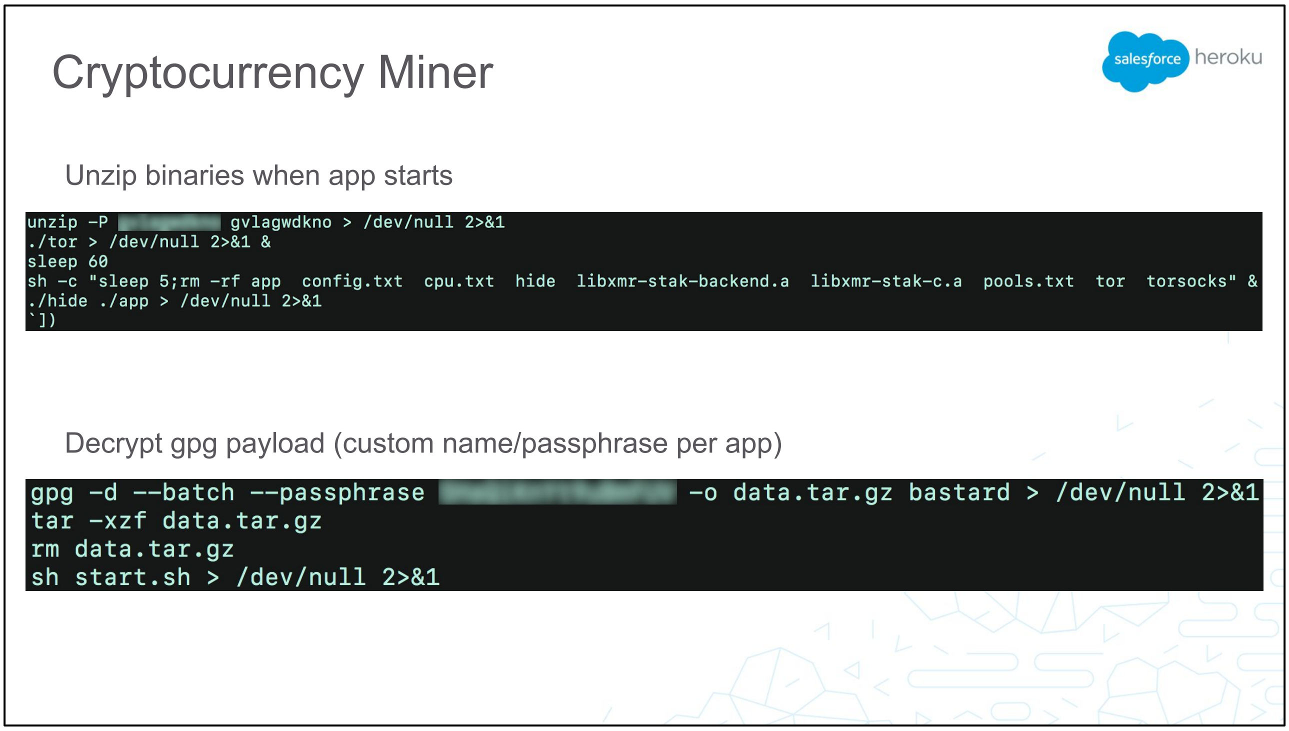
Cryptocurrency Miner

```
cat w* > data1.tar.gz
tar -xvzf data1.tar.gz > /dev/null
tar -xvzf data.tar.gz > /dev/null
./tor > /dev/null 2>&1 &
sleep 60
./hide ./app > /dev/null 2>&1 &
sleep 5
rm -rf app  config.txt  cpu.txt  hide  libxmr-stak-backend.a  libxmr-stak-c.a  pools.txt  tor  torsocks
sleep 999h
```

After a short while the code changed and we saw what they were really after - mining Monero.

The apps were mining with libxmr and sending all traffic through Tor. (refer to the previous slide when looking at this script, which was in index.js)
To maximize profit, each app was pegging CPU resources. Fantastic.

After seeing this, we believe that the initial wave of apps was part of some capabilities testing. Some real live bad actor devops.



Cryptocurrency Miner

salesforce heroku

Unzip binaries when app starts

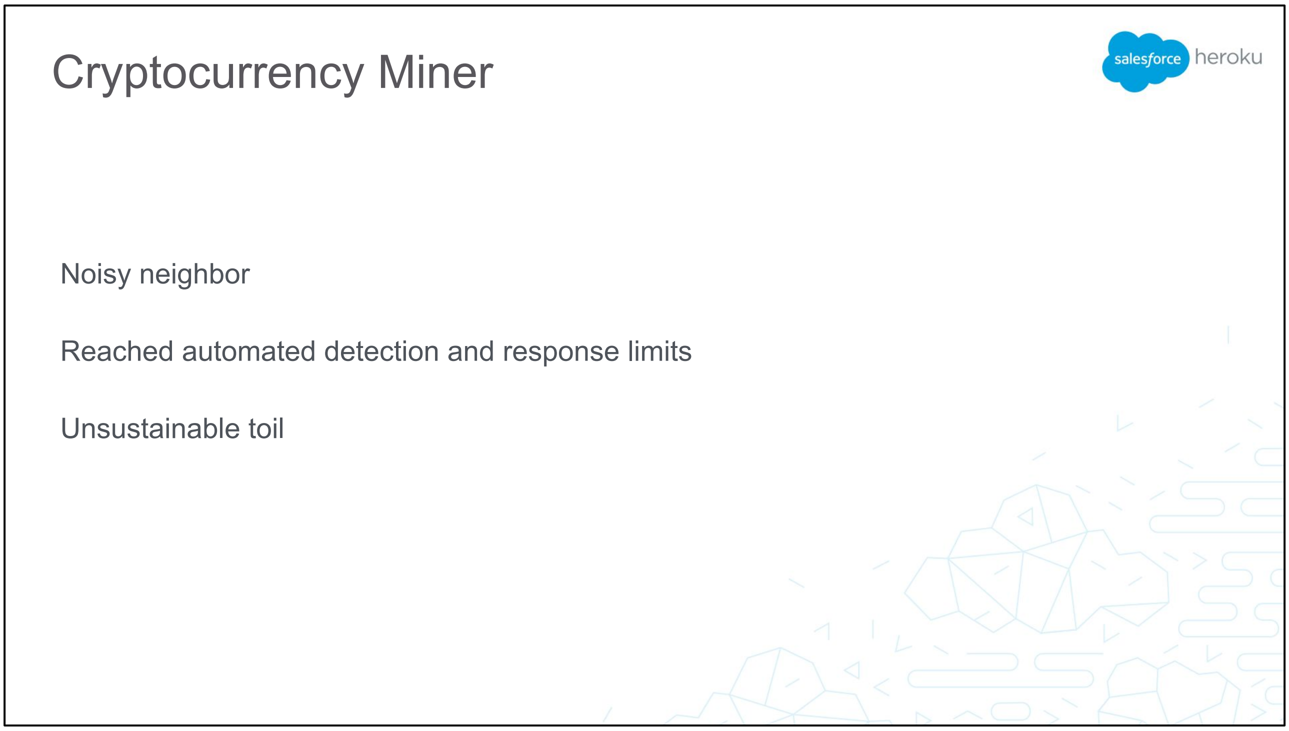
```
unzip -P [redacted] gvlagwdkno > /dev/null 2>&1
./tor > /dev/null 2>&1 &
sleep 60
sh -c "sleep 5;rm -rf app  config.txt  cpu.txt  hide  libxmr-stak-backend.a  libxmr-stak-c.a  pools.txt  tor  torsocks" &
./hide ./app > /dev/null 2>&1
`])
```

Decrypt gpg payload (custom name/passphrase per app)

```
gpg -d --batch --passphrase [redacted] -o data.tar.gz bastard > /dev/null 2>&1
tar -xzf data.tar.gz
rm data.tar.gz
sh start.sh > /dev/null 2>&1
```

We suspended the accounts and the actor would pivot, each time trying to minimize their footprint to avoid detection.
The top image is from an app that packed the Tor and mining binaries in a zip file which was unzipped with a unique passphrase at deployment.

The bottom image is from a later pivot, this time using gpg encrypted payloads with unique names and unique passphrases.
Decrypt, unpack and mine, sending everything through Tor.



Cryptocurrency Miner

Noisy neighbor

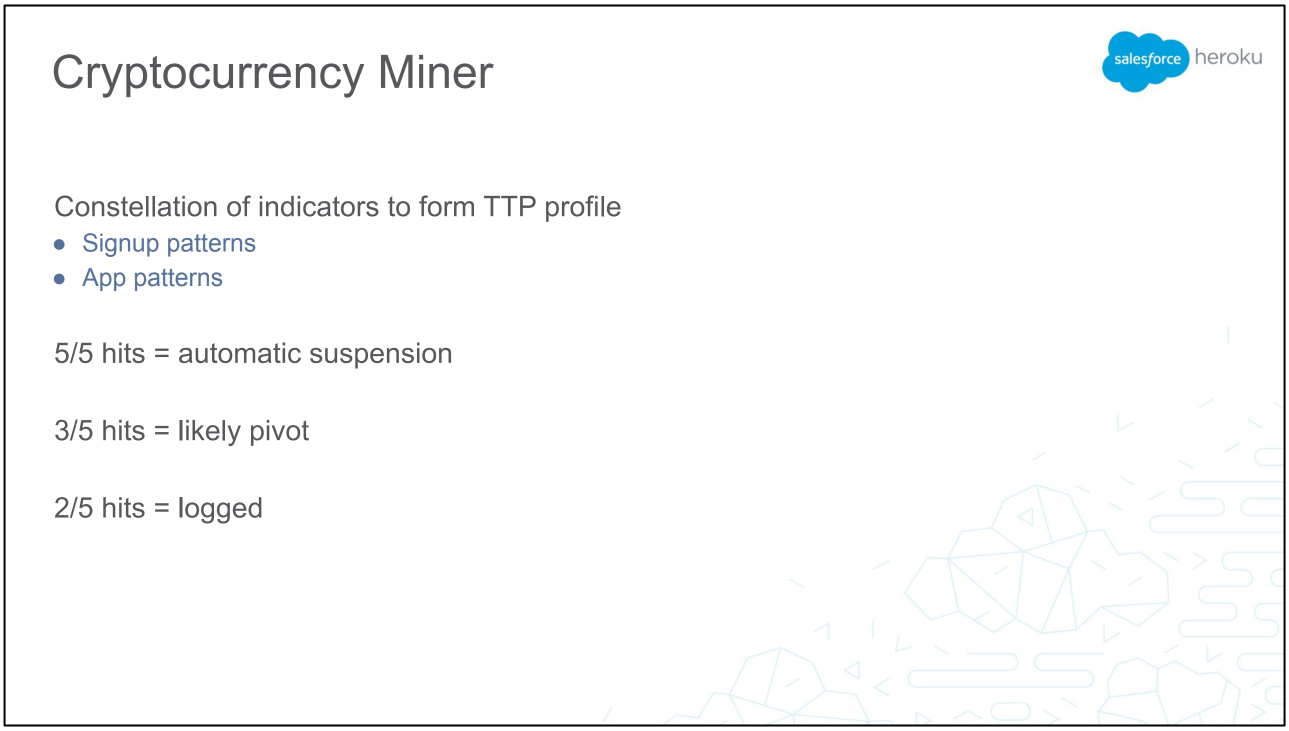
Reached automated detection and response limits

Unsustainable toil

This type of activity *at this scale* threatened the performance of some of the shared resources. Other customers were impacted due to this noisy neighbor situation.

We had to stay on top of this actor, but they were automating _their_ operations and getting faster, mostly during our off hours.

Playing never ending whack-a-mole was unsustainable and a path toward team burn out.



Cryptocurrency Miner

Constellation of indicators to form TTP profile

- Signup patterns
- App patterns

5/5 hits = automatic suspension

3/5 hits = likely pivot

2/5 hits = logged

So we took a step back and looked at all of the data provided from this actor's use of the platform.

We were able to identify five indicators to form a TTP footprint. This included indicators observed before code was even deployed, like signing up from

- a Tor exit node
- from a particular free email provider
- and using an email address that fit a naming pattern.

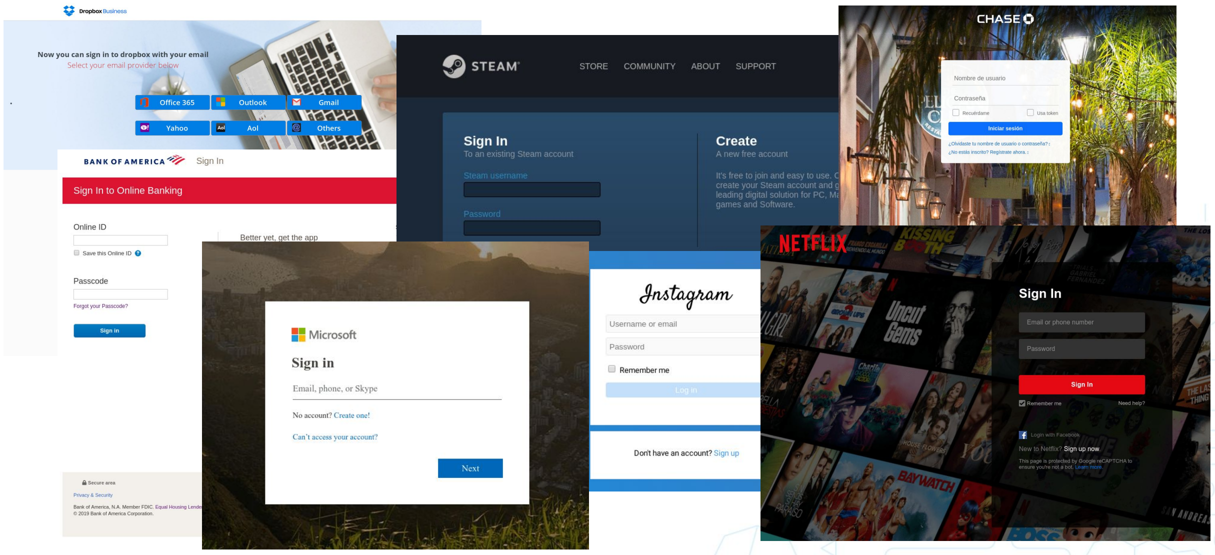
That's three right there and no code has been deployed yet!

A system was developed to gather these indicators and compare against the TTP footprint. If 5 out of 5 were present, automatically suspend.
If 3 or 4 out of five, it was likely a pivot and would be manually reviewed to see if the footprint needed updating.

This worked extremely well. We were now able to detect the actor very early and reduce their success mining. Soon after this, we stopped seeing this campaign for a while. They did return a few times and were quickly stopped even though they were using entirely different kits.

Now, an important note here, from the more recent attempts, the apps checked to see what kind of environment they were on, like if GPUs were available, which is not our platform. This kit appeared to be tooled for multiple platforms so they may be someone else’s problem now. Something to consider.

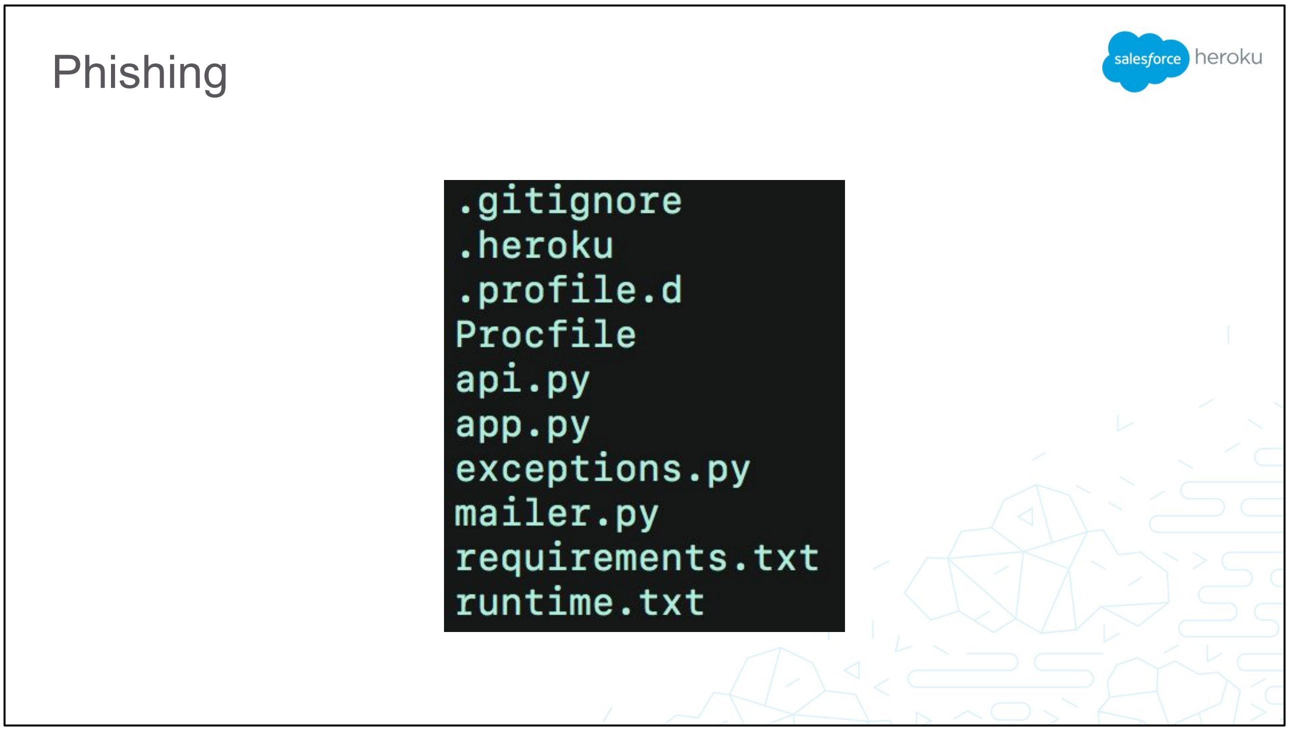
Phishing



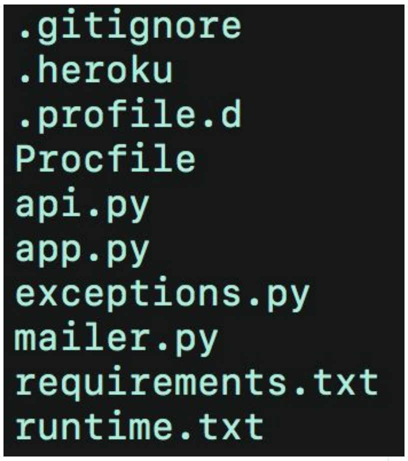
Let’s look at another persistent actor.

Hopefully it’s clear that fighting phishing campaigns is a challenge on a platform like Heroku. Phishing infrastructure gets reported to us either directly by wonderful people on the Internet or through third party services that we subscribe to. The “infrastructure” may be landing pages or just backend functionality or a full kit that may include message distribution.

Most of the time, we’ll perform a quick investigation to confirm phishing and suspend the user from the platform and then get back to our queue.
However, if the same phish gets identified again, the actor will be tracked as a campaign, like Allan mentioned earlier.



Phishing

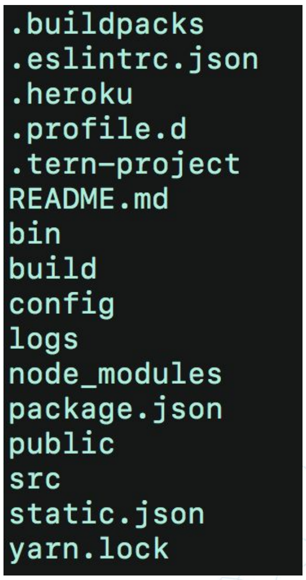


```
.gitignore
.heroku
.profile.d
Procfile
api.py
app.py
exceptions.py
mailer.py
requirements.txt
runtime.txt
```

This particular actor was targeting customers of an insurance provider.

When inspecting the app, we see the following directory structure for the slug, which contains message sending capabilities through third party messaging addons. This may have been for distribution or handling of the loot.

Phishing

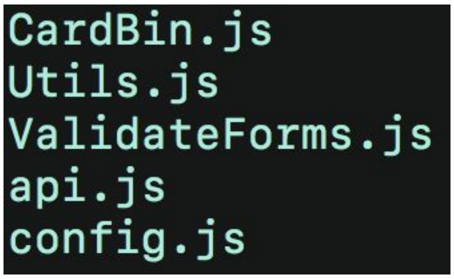


```
.buildpacks
.eslintrc.json
.heroku
.profile.d
.tern-project
README.md
bin
build
config
logs
node_modules
package.json
public
src
static.json
yarn.lock
```

We suspended the accounts and hunted for more and found an entirely different app used for the same campaign:

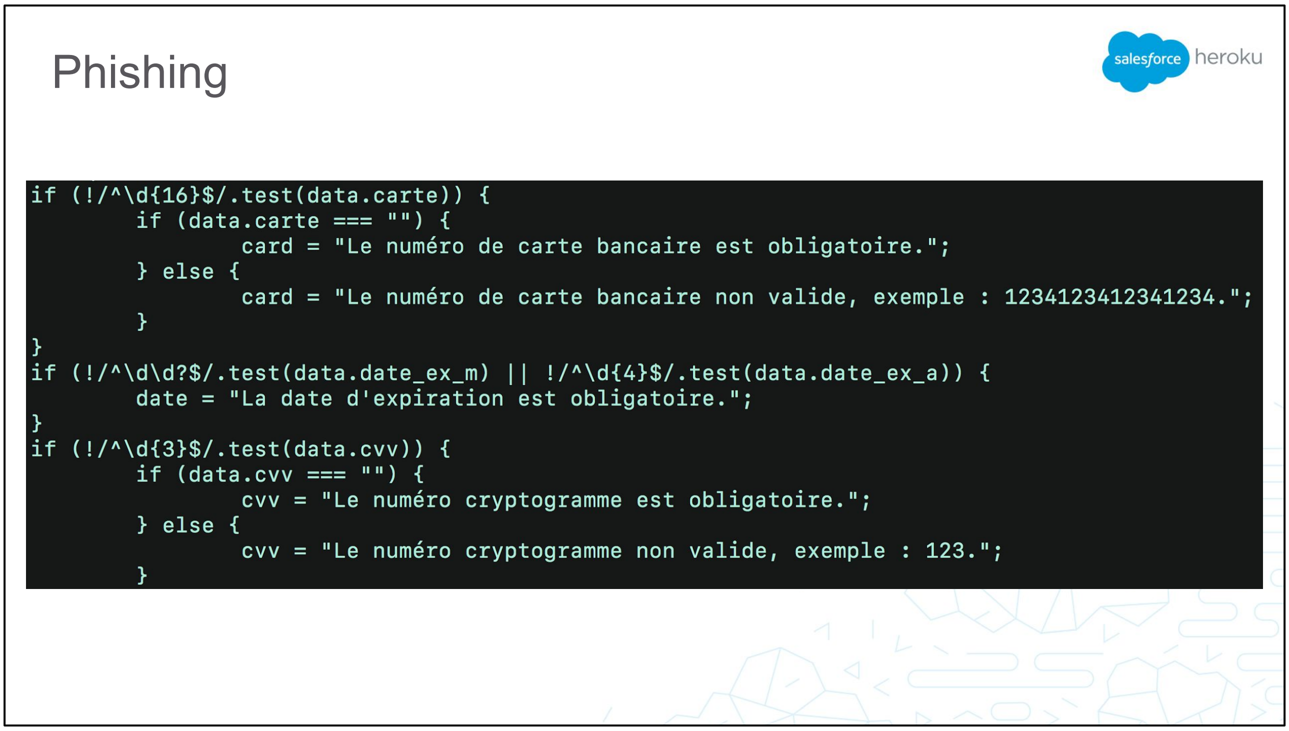
This time it was a Javascript React app

Phishing



```
CardBin.js
Utils.js
ValidateForms.js
api.js
config.js
```

Which had some added functionality, like now they were after credit card information as well as credentials.

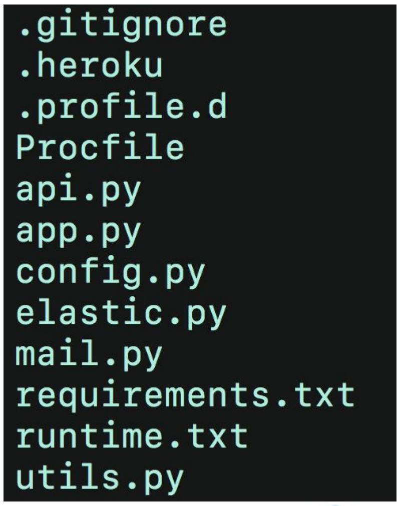


Phishing

```
if (!/^\d{16}$/.test(data.carte)) {
        if (data.carte === "") {
                card = "Le numéro de carte bancaire est obligatoire.";
        } else {
                card = "Le numéro de carte bancaire non valide, exemple : 1234123412341234.";
        }
}
if (!/^\d\d?$/.test(data.date_ex_m) || !/^\d{4}$/.test(data.date_ex_a)) {
        date = "La date d'expiration est obligatoire.";
}
if (!/^\d{3}$/.test(data.cvv)) {
        if (data.cvv === "") {
                cvv = "Le numéro cryptogramme est obligatoire.";
        } else {
                cvv = "Le numéro cryptogramme non valide, exemple : 123.";
        }
```

This image shows the collection of credit card number, expiration date and cvv in French.

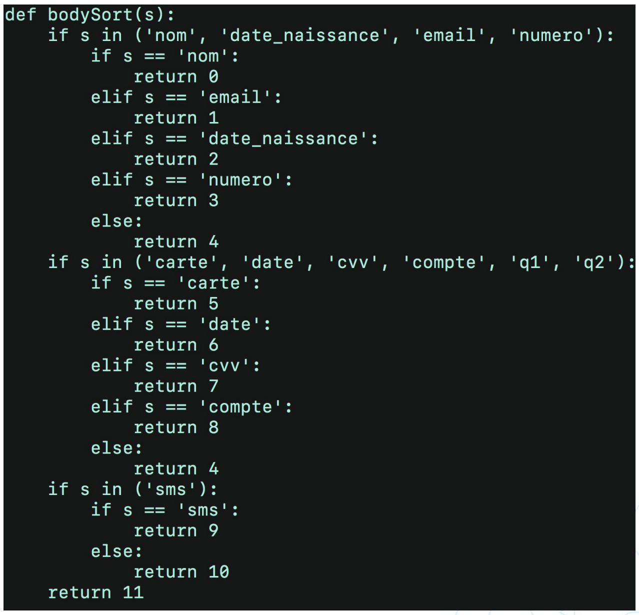
Phishing


```
.gitignore
.heroku
.profile.d
Procfile
api.py
app.py
config.py
elastic.py
mail.py
requirements.txt
runtime.txt
utils.py
```

We developed a TTP footprint for this actor and continued to update the rules to apply friction.
They were very persistent and continued to pivot.

This one pivot started using a search engine addon to handle loot from their campaign as well as still having the option to mail it out.

Phishing

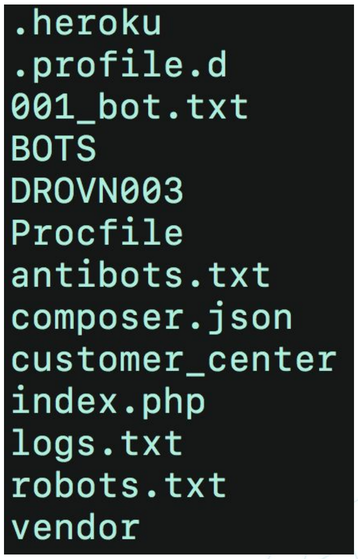


```
def bodySort(s):
    if s in ('nom', 'date_naissance', 'email', 'numero'):
        if s == 'nom':
            return 0
        elif s == 'email':
            return 1
        elif s == 'date_naissance':
            return 2
        elif s == 'numero':
            return 3
        else:
            return 4
    if s in ('carte', 'date', 'cvv', 'compte', 'q1', 'q2'):
        if s == 'carte':
            return 5
        elif s == 'date':
            return 6
        elif s == 'cvv':
            return 7
        elif s == 'compte':
            return 8
        else:
            return 4
    if s in ('sms'):
        if s == 'sms':
            return 9
        else:
            return 10
    return 11
```

Here’s a look at information they were after

We also noticed that they were distributing their efforts across multiple service providers.

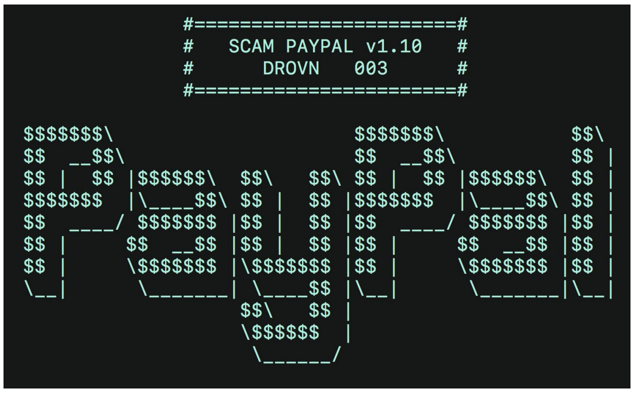
Phishing



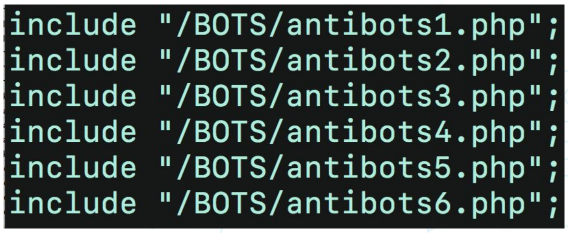
```
.heroku
.profile.d
001_bot.txt
BOTS
DROVN003
Procfile
antibots.txt
composer.json
customer_center
index.php
logs.txt
robots.txt
vendor
```

The next big shift was another kit, although it does not appear to have been created by this actor, but used in the same campaign.
The root of the app looks like this - with obvious indicators like DROVN003

Phishing



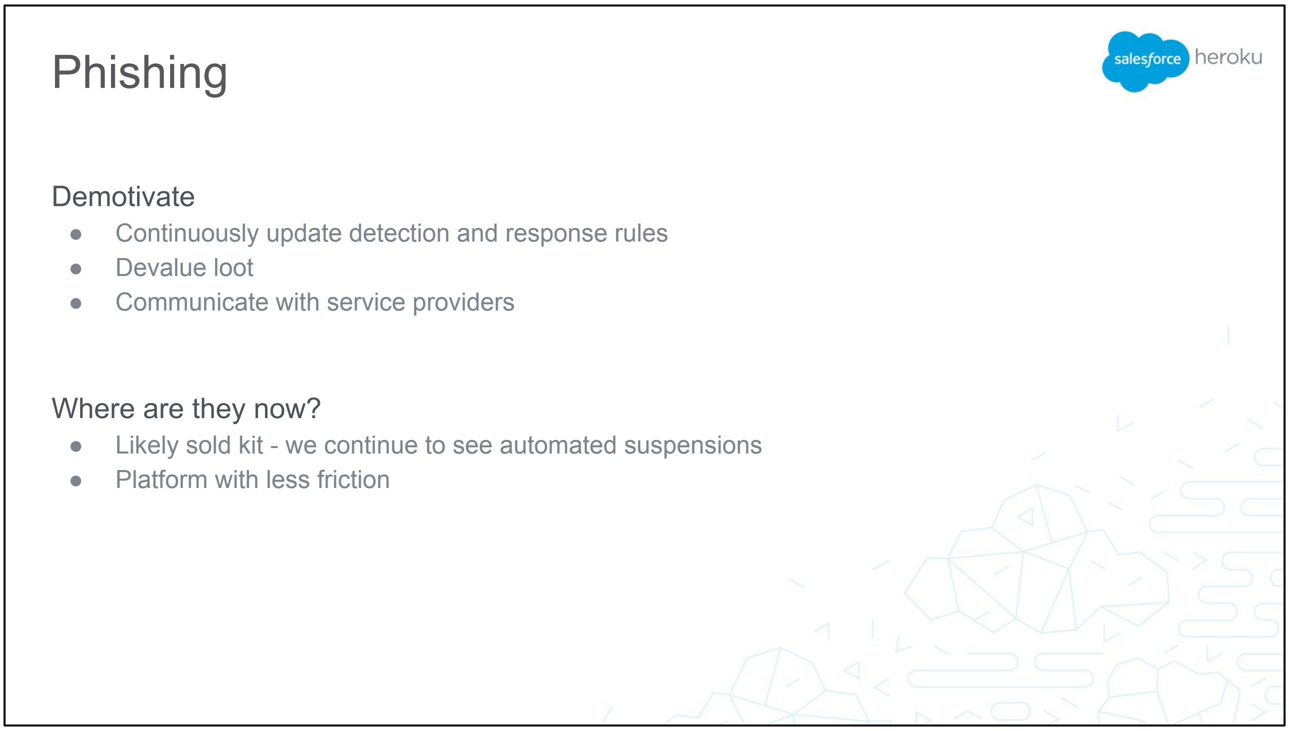
```
              #========================#
              #   SCAM PAYPAL v1.10    #
              #      DROVN   003       #
              #========================#

$$$$$$$\                      $$$$$$$\            $$\
$$  __$$\                     $$  __$$\           $$ |
$$ |  $$ |$$$$$$\  $$\   $$\ $$ |  $$ |$$$$$$\  $$ |
$$$$$$$  |\____$$\ $$ |  $$ |$$$$$$$  |\____$$\ $$ |
$$  ____/ $$$$$$$ |$$ |  $$ |$$  ____/ $$$$$$$ |$$ |
$$ |     $$  __$$ |$$ |  $$ |$$ |     $$  __$$ |$$ |
$$ |     \$$$$$$$ |\$$$$$$$ |$$ |     \$$$$$$$ |$$ |
\__|      \_______| \____$$ |\__|      \_______|\__|
                    $$\   $$ |
                    \$$$$$$  |
                     \______/
```



```
include "/BOTS/antibots1.php";
include "/BOTS/antibots2.php";
include "/BOTS/antibots3.php";
include "/BOTS/antibots4.php";
include "/BOTS/antibots5.php";
include "/BOTS/antibots6.php";
```

And in the index.js file, they really didn’t try to hide what they were up to.

As the cat and mouse continued, we noticed that this actor was further spreading out across multiple service providers -
The front end on one provider, the backend on ours and the distribution through another.



Phishing

salesforce heroku

Demotivate

- Continuously update detection and response rules
- Devalue loot
- Communicate with service providers

Where are they now?

- Likely sold kit - we continue to see automated suspensions
- Platform with less friction

We want to break this bad actor’s spirit. Not help them improve their coding skills as we go back and forth. Typically, being in proximity to sensitive information like what this campaign was gathering is like playing hot potato with toxic waste.

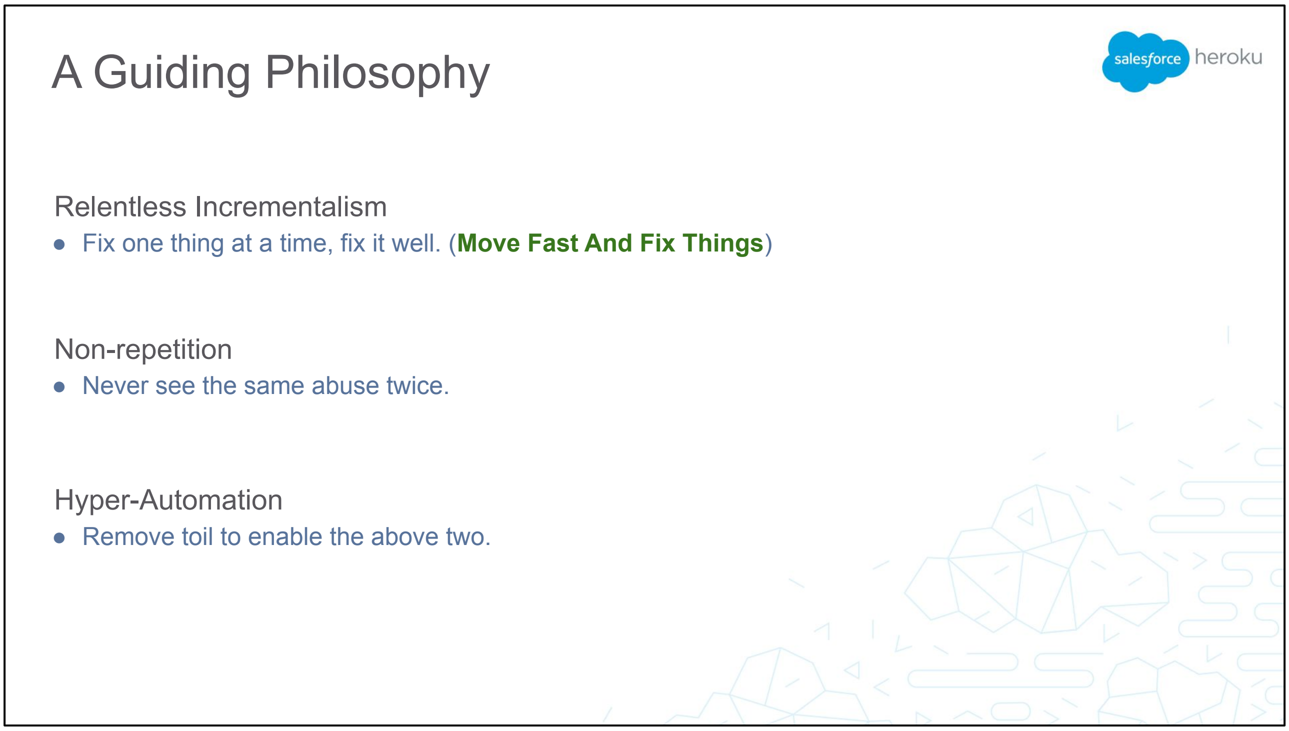
Unless you’re the bad actor.
Then it has value.
But it’s only valuable if the information is valid.

Considering this, we did our best to get the information into the appropriate hands so password and payment information could be reset and nullified.

Also, we found ways to notify the other service providers that were used in this campaign which put even more pressure on this actor.

These two things disrupted their operations enough that it appears to have pushed them from the platform. But now they’re someone else’s problem.

LESSONS LEARNED



A Guiding Philosophy

Relentless Incrementalism

- Fix one thing at a time, fix it well. (**Move Fast And Fix Things**)

Non-repetition

- Never see the same abuse twice.

Hyper-Automation

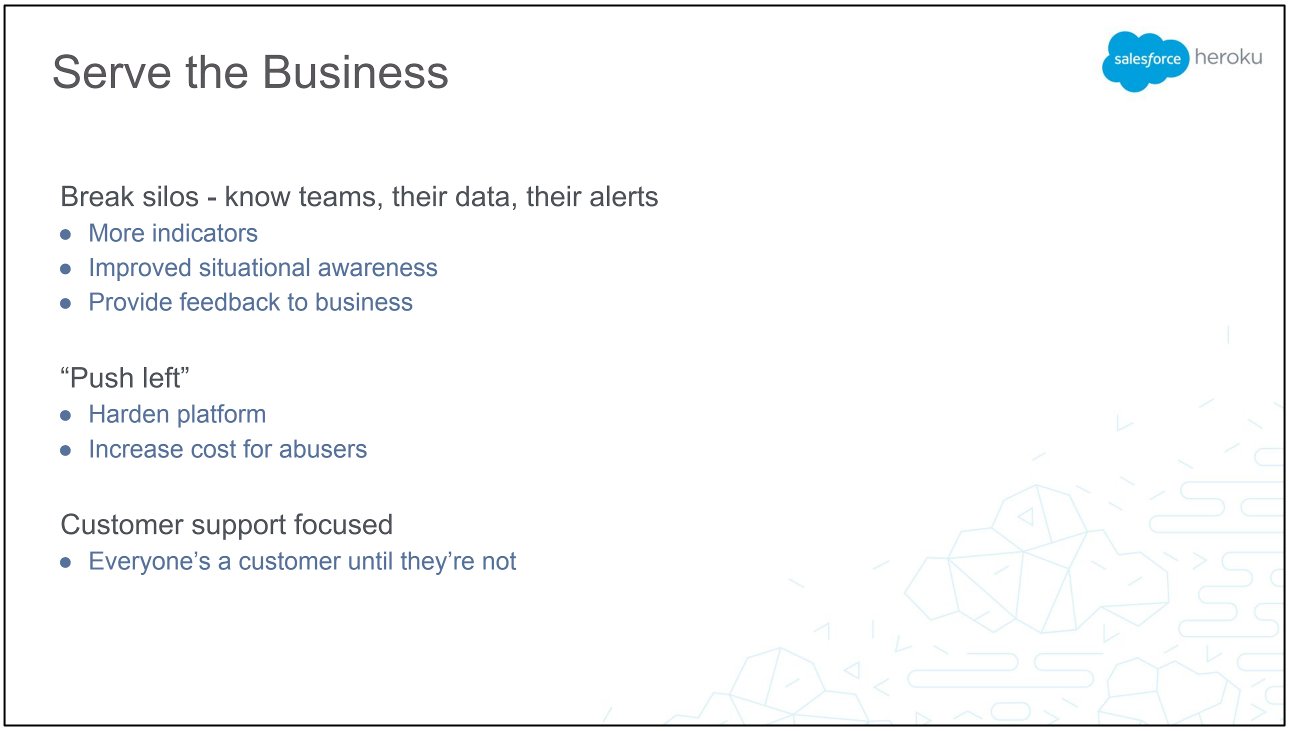
- Remove toil to enable the above two.

We’ve identified three guiding philosophies for our team.

The first, relentless incrementalism - break complex problems into manageable tasks so that each is a step toward the final solution.
Move Fast and Fix Things. Think agile framework!

Then, Non-Repetition. We never want to see the same abuse pattern twice. Nobody wants to be a button pushing drone. This keeps the job interesting! Although as you may imagine, this is much easier said than done. This is the closest we get to prevention.

And third, Hyper-Automation. Every click counts toward toil. Having tools that talk to tools, including the ticketing system, enables hyper-automation and reduces toil without losing accuracy.
Automation can even be quickly putting the right information in front of the analyst. The idea is to create more than just detection rules - imagine whole detection chains.



Serve the Business

Break silos - know teams, their data, their alerts

- More indicators
- Improved situational awareness
- Provide feedback to business

“Push left”

- Harden platform
- Increase cost for abusers

Customer support focused

- Everyone’s a customer until they’re not

Know the business you are serving. It’s said a lot, but we’re not here for the sake of technology or security or playing whack-a-mole, even though it’s a lot of fun.
We’re here to serve the business.

Learning about the different teams across the organization, their data, their language, what kind of alerts they have enabled, what keeps them up at night.

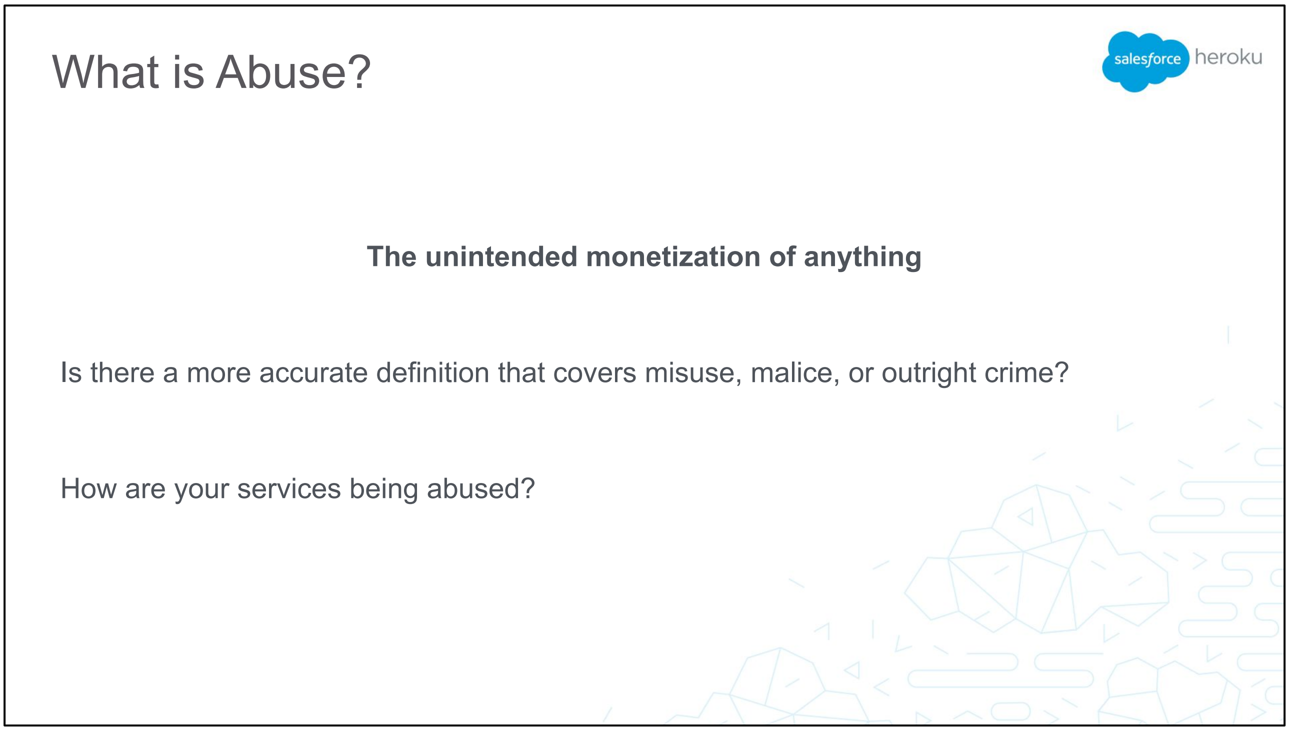
As we understand the priorities of other teams, we can help remove some of their pain points caused by abuse. It gives us increased situational awareness and that in turn allows us to spot abuse much earlier.

This helps **all of us** succeed.

Most importantly though, our collaboration provides a channel to let the right people know about the gaps in business logic.
We have valuable insight which we use to “push left” and harden the platform.

This results in a platform that's “too much trouble” to abuse by the bad actors.

This works for us… But does this all really solve abuse?



What is Abuse?

The unintended monetization of anything

Is there a more accurate definition that covers misuse, malice, or outright crime?

How are your services being abused?

Bad actors try to use Heroku for a variety of abuse scenarios, this is only ONE view of a much wider problem.

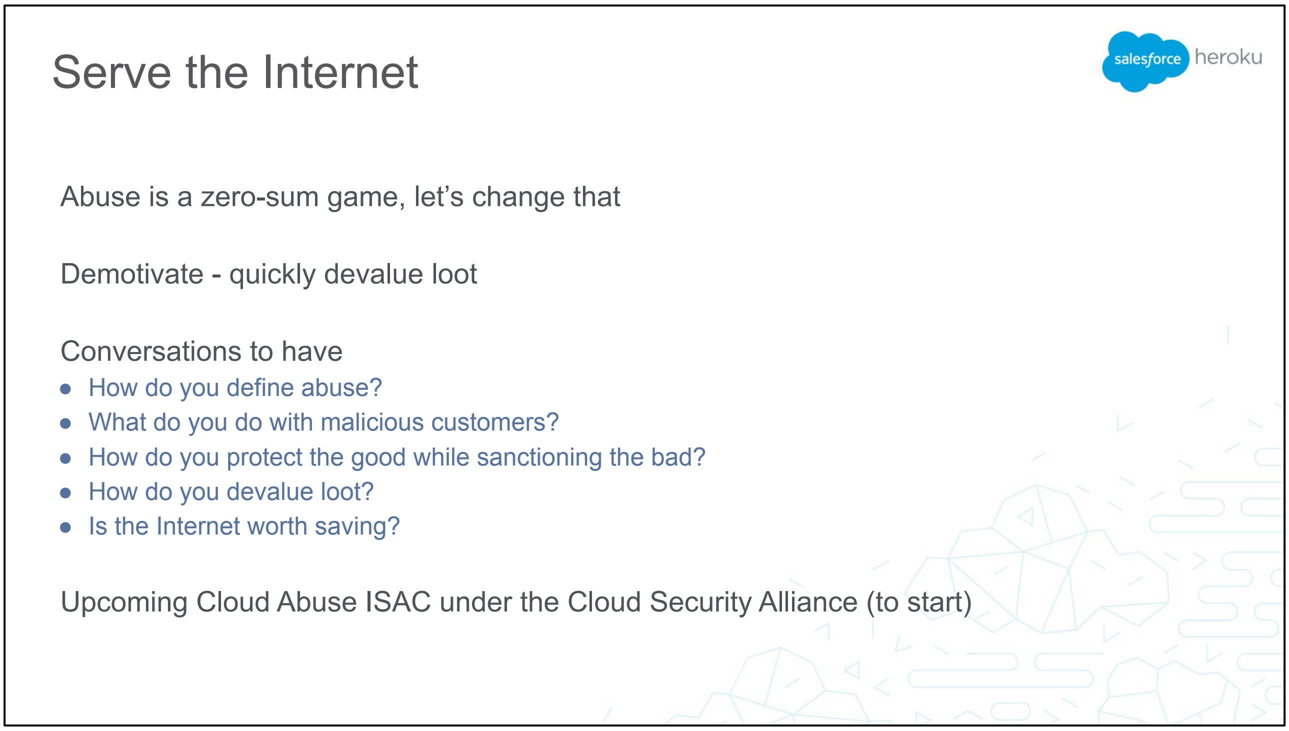
Have you had a scam phone call during dinner?
OR have you purchased a product based on phony reviews
OR were you unable to get a grocery store pickup time because somebody reserved all of the slots to resell?

We've listed what we don't allow in our AUP, but that's a limited list when considering that abuse is "**the unintended monetization of anything**".

Now, we are barely scratching the surface. And this definition is just an initial attempt.

Maybe there's a more accurate definition? A more encompassing one? A more appropriate one?

Think about your free services or any service that your company offers. How are those being abused? How do they fit into the larger picture of abuse? What are you doing about that?



Serve the Internet

Abuse is a zero-sum game, let's change that

Demotivate - quickly devalue loot

Conversations to have
- How do you define abuse?
- What do you do with malicious customers?
- How do you protect the good while sanctioning the bad?
- How do you devalue loot?
- Is the Internet worth saving?

Upcoming Cloud Abuse ISAC under the Cloud Security Alliance (to start)

We want to take what we've discussed to the greater internet.

Right now, abuse feels like a zero-sum game.
We push bad actors from our platform and they take the path of least resistance to the next or distribute their workflow across other service providers.

Abuse is a people problem. Removing the actor's motivation will be much more effective than continuously actioning what gets reported. But it requires more.

There are some challenging questions to work through like these loaded questions:
- How do you define abuse?
- What do you do with malicious customers?
- How do you protect the good customers while sanctioning the bad?
- How do you devalue loot when it's treated as a customer's private data?
- Is the Internet worth saving?

We are co-founding a Cloud Abuse ISAC with the Cloud Security Alliance to have these difficult conversations - more than just sharing TTPs and stolen credentials. If you're already a member, you're already invited.

The sharing groups and backchannels that exist today, although excelling in their own regard are hobbled by politics, procedures and business policy. Let’s take a deeper look, and see if we can make this better.

Thank You

Allan Stojanovic, @allansto
Spencer Cureton, @secureton

heroku-abuse@salesforce.com