CLOUDBLEED CASE STUDY

By Cynthia Alise Mock

Introduction

In February of 2017, a bug in Cloudflare's services was discovered. The bug when triggered by broken html, leaked information from buffers that could contain personal information (Holland, 2017). There was a shockwave in the consumer confidence of providers of that type of service. Bug sometimes revealed personal information at random making consumers think that visiting websites on the cloud was not secure. The impact that this had on the distribution of content was that webpages did not load effectively. The random nature of the disclosure of information meant that the damage was less than if it were an intentional virus, even though it was discovered months after it began to occur. This Case Study is a look into the technical and security issues around solving the problem of the bug.

The worst part of the bug is that it could reveal personal information that was stored in the buffers (Wikipedia, 2017). Everything from messages, to locations could have been dumped into a viewable form on the internet. This meant that there was a breach in the confidentiality of sites such as OK Cupid and Uber. The kind of information stored in their buffers could aid in cybercrime and cyberstalking. The only advantage of the dump is that the bug dumped information at random without any order, so it was not efficient for hackers to use as a means of gathering information. You had to understand the code and be lucky to use the display.

The bleed effected the display of information from a website and therefore the content delivery. From the point of a broken html tag, the buffer pushed information from an adjacent buffer displaying what was saved there. This caused the information to be exposed and the rest of the page not to be displayed. This meant that unjustified code prevented the delivery of

content. Because the bug needed a broken tag to display, the websites that triggered it were not major sites with a lot of traffic. This had an effect on the websites ability to serve its customers when the page was incomplete. This caused a loss in business and trust.

The personal information that was revealed appeared at random and could contain any information from the buffer. This could include anything from dating messages to your most recent Uber location. It basically posted to the web any information in the buffer it bleeds into. The bug was unintentional, therefor there was no control on what was or was not disclosed.

The problem wasn't discovered for a while. It was believed that the bug began September 2016 and was fixed in February 2017 (Prince, 2017). Immediately upon discovery, Cloudflare began testing to assess the damage and asserted that it was not the same as a traditional data breach. What could have been leaked during that period is unforeseeable. However, efforts to make Cloudflare more secure are already under way.

The bug effected major services such as OK Cupid, Fitbit and Uber, but testing revealed that the bug probably was unintentional. Testing was done to assess how likely it was that the bug had been exploited by malicious hackers and the conclusion is that there is little evidence to support that the bug was efficient for that type of use (Prince, 2017). Cloudflare published the results of their assessment on their website and expressed that the damage had been minimal because hackers were not aware that they could exploit the problem. They assessed access statistics and stated that there was not a lot of recorded suspicious activity related to the bug. However, some of the time the bug presented no meaningful statistics exist.

There are potentially several solutions for the problem one is to program a stop when Cloudflare encounters broken HTML, another is to increase the buffer size for a page and finally,

to program something that would prevent other buffers from being used with adjacent pages. This case study surmises that a stop in reading a broken page is critical to the health of this cloud service. Computers deal in finite functions and having it recognize cloud data as finite is healthy for the use of its services.

Background

Cloudflare is a cloud based service that helps to provide distributed content systems on the web. The service is used by major hitters in the industry like OK Cupid, Uber, and Fitbit to increase the speed of their sites. The distribution systems used buffers that were getting over run when they encountered errors in the HTML. What was returned on the screen was information from an adjacent buffer that could potentially contain sensitive data. The page would end and displayed below was the buffered information. The information returned was random, however could sometimes return personal information like messages and credit card numbers. Cloudbleed as the bug was termed was not maliciously planted in the Cloudflare system. It was randomly generated from an error in programing. It was triggered under certain circumstances. Cloudflare underwent several tests to find out if the error was maliciously exploited and deemed that there was no evidence that the error was being used intentionally. Smaller unfrequently visited sites were the most venerable to the Cloudbleed error. Major sites had less html issues and were less likely to trigger the bug, and the leaks occurred on less frequented sites. Major sites had less broken code.

Proposed Solution

A proposed solution for the Cloudbleed bug is to create a stop in the reading of the HTML when it is broken so that it doesn't pull information onto the page that isn't meant to be there. Programming this feature into the Cloudflare system will fix the bug and allow it to spare precious resources on top of solving the problem. The broken code can serve as a kind of signature that the parser can recognize and account for. Treating the code as if it had a virus that needed not to be executed would stop the bleed from occurring. The downside to this solution is that the content that is below the error will not be displayed, however there also will not be a leak of information. The ability to spare the buffers and save space on encountering an error will help the system run more smoothly in the long run. For the sake of functionality this is my preference for solving this problem because of the efficient use of space and resources.

This solution puts a burden on web designers to make sure that their pages are coded correctly, with out creating leaks when they are not. Often content management systems like Wordpress create the code independently of the author of the page. In situations were there is little control over the coding, making sure that a website validates would be difficult. This might be part of what is causing the triggering event. Making sure that the site is free of broken code is a challenge for modern day web designers. The solution of just not letting the page display would fix problems with these anomalies so that the damage would be minimal.

The advantage to this system is that when the page stops displaying and does not bleed into the buffer, the resources of the buffer are spared, and the information contained in the adjacent buffers would remain there. This would lead to increased security and a fail safe for broken code. The makers of the page could test it and would be challenged to make their sites code more complete.

Alternatives

1) The first alternative is to increase the buffer size of the pages in question so that they do not over run into adjacent channels. This solution could be full of errors since it is difficult to estimate the amount of buffering that a page is going to need and the fact that this would tax the system in a way that might keep it from being able to deliver its content. It would seem that this alternative is not as preferable as the one chosen. Stopping the page rather than trying estimate a larger buffer would be more practical and lead to better performance of the system. The only upside to this alternative is that the page will still display beneath the broken html.
2) The second alternative is to create a stronger barrier between the buffers. This alternative could be used in conjunction with the solution chosen. Reinforcing the buffers so that they do not seek adjacent buffers, would prevent the leaks. This however, does not enhance performance as much as the solution chosen. The solution chosen will mean that loading pages that are broken get abandoned more quickly and the system is less burdened by the bad pages errors.

Recommendations

To implement the proposed solution, code needs to be added to the servers that handle the buffers. This is no small suggestion since the services that Cloudflare provide are substantial. A team of server experts need to modify the code and upload it to all Cloudflare's servers. This could be done asynchronously to preserve service during the update. The advantage to the decentralized structure of Cloudflare is that updates can be done one server at a time, while the traffic is rerouted to servers that haven't been updated yet. There are two parts that need to be completed. The creation of the new server code and a plan for the update. This can be accomplished on staff engineers and support staff. Running the update can potentially be

automated to reduce the amount of staff needed for the roll out. Creating code that will account for the broken html should not be too difficult for Cloudflare's staff. If there are resources there are resources to delve further into the problem, then alternative two could also be implemented in the same fashion.

Finalizing the Case

In conclusion, the Cloudbleed bug, although unintentional, poses a serious security risk. Recommending in this case study are steps that can prevent this from occurring on Cloudflare's servers, better protecting those that use its services. The decentralized nature of Cloudflare means that the updates are possible without outages and that preventative measures should be carried out. They are accomplishable and discrete. I propose one solution in the case study, however the alternatives can be used in combination with each other to better reinforce the programming on the Cloudflare server. In totality this problem is preventable in the future and the random nature of the error makes the data breach less critical.

References

Wikipedia. (2017). *Cloudbleed*. Retrieved on December 17, 2017 from: https://en.wikipedia.org/wiki/Cloudbleed

Prince, M. (2017). Quantifying the impact of "cloudbleed." Retrieved on December 16, 2017 from: https://blog.cloudflare.com/quantifying-the-impact-of-cloudbleed/?_bt=183543305982&_bk=cloudbleed&_bm=e&_bn=g&gclid=CjwKCAiA693RBRAwEiwALCc3u6FqU3bQeEqCwKeqeneFG3MRJxVIsVmKlXECF62kohMpaiDODUn-9BoCJAkQAvD_BwE

Holland, P. (2017). Cloudbleed: everything you need to know. Retrieved on December 18, 2017 from: https://www.cnet.com/how-to/cloudbleed-bug-everything-you-need-to-know/