Security in Startups

Michael B. James
hello@michaelbjames.com

Mentor: Aaron Boyd
aaron@nomic.com

December 15, 2013

Abstract

By devoting all of their time to making a product, startups rarely take the time to step back and review the security of their product. Creating a minimum viable product always comes first but the business costs of getting abused from a simple security hole often outweigh that amount of time spent creating a new feature. Using new technologies and full of programmers aware of SQL injections and XSS, they often overlook other security holes. I will outline the forgotten parts of security in startup life, their impact, and how they are used maliciously. I will also suggest preventative, non-disruptive measures that are not time consuming so that a startups product can maintain security throughout the development process.

1 Introduction

Before making a storefront safe from would-be thieves, there first needs to be something worth stealing. Startup companies follow this same pattern albeit from a different kind of would-be thief. The potential threat does not walk in the front door, though one should protect that too[1], the would be attacker goes through already open doors–server ports. A small company needs a new and interesting product before starting to grow up (e.g., Instagram or Snapchat). It's obvious that engineering, product promoting, and development all have to be a startup's mission. However what is not entirely clear is when a company should start locking down their app, service, or website to prevent something catastrophic.

It comes with no surprise that the people who work in tech companies are young[7]. The median programmer at a company has not yet seen his or her 10-year college reunion. This is to say that the average worker knows newer and hipper languages such as Python, Ruby, and Javascript. Furthermore this newer generation is more aware of previous-era security holes such as SQL-injection. Moreover, with injection attacks having been a prevalent problem for quite some time, there are dozens of wrappers that make database queries safer in newer languages. This means that it is often times the other common holes that end up causing problems. In other words, it is the sum of the less frequent holes that is greater than the most frequent source of security concerns.

The first common flaw in a new company trying to move quickly is to overlook simple logical problems in the system that has been created. There is plenty of pressure on developers to overcome bugs and add features and as a result what has already been written is not given the same kind of proofreading a magazine or book would get. The underlying problem is one of ensuring that the logic of the system is sound. This is to say that the user cannot accomplish a feat that was intended to be possible (e.g., changing another user's password).

Secondly, there are a variety of different kinds of modules in any software stack. They may run on shared or different platforms. In any case, these are not always kept up to date. It is too easy to install a specific version of software and forget about it entirely. Every so often a hole is found in practically universal server hosting software (e.g., nginx or apache). The safety of the base layer is paramount to the safety of the layers sitting on top of it. The further down a security bug is discovered the more damaging it can be.

The third problem is not with the code but with the writers of that code. The average developer in a startup is relatively inexperienced mostly due to age. Not having seen more than a decade of production code's success and failure puts the recent college graduate at a security disadvantage. The new police officer is not privy to many evasion tactics but the old police officer can spot 'a runner' at first glance. In the same way, a new programmer does not consider every way in which a piece of code could break but a developer who has witnessed code utterly fail thinks about new ways his or her code could break.

2 To the Community

2.1 Something Catastrophic

Overlooking these security holes could at best go totally ignored. At worst, the service has to shut down (e.g., Lavabit[6]–although the security hole is the NSA in this case). The middle ground is not terribly appealing either.

When firesheep[3] came out in 2010, websites reacted by shaping up and serving much of their page over SSL. No user would want to access a site in a coffee shop and let his or her session get taken over. Trust in the service would plummet as fast as the bitcoin on a bad day. One now-famous startup was sending sessions and more importantly login information in the clear. Later that same year, Instagram was

caught letting user passwords go over the air on their iPhone app[4]. This was horrible: many users repeat passwords across services (while it is bad practice, it is ubiquitous) so by letting their Instagram account get compromised, everything else gets compromised.

Every service has its share of curious users, who want to poke the service is interesting ways and see what happens. Snapchat's service is to make pictures and videos ephemeral. However, through simple short-sightedness, the android developers forgot about where the service caches its media[2]. As a result, anyone with an android phone–it did not need to be modified–could save the media. For some users then, Snapchat is immediately ruled out as a safe way to share silly photos with friends. That signifies an entry barrier for some users. There is something to be said with the social heft the service carries, as of 2013 it is difficult to find an alternative that one's friends also use. However, this still is a sign of not being thorough with the app-testing.

Sometimes it is a problem with the actual service. In a previous version of Nomic's server API there was no verification that a user was who they said they were with certain API calls. As a result, a specific API call could change any other user's password to a specific value. If Nomic had a larger user base before I discovered the bug and someone else found that flaw, the results could have been catastrophic. Everyone's password could have been changed, locking everyone out. This would have necessitated a password change of everyone in the system. The awkward situation was prevented by simply checking over the API route rules (see attached code for example).

These first three examples could have been prevented entirely by giving the code base another look over. None of these were special cases of the program running with certain values in memory that cause a stack overflow and allow code injection. These are not bugs that require much time to test. These three are logic oversights. Startups need to get features out quickly but in moving too quickly, certain assumptions are made that are not always true (i.e., mobile web traffic is safe, the cache is hidden, only the account owner will try to change their own password).

In 2011 the Play Station Network, a fairly new service by Sony, was faced with an unfortunate situation. Their service uses SSH for administrative and development purposes. However, they were using a version five years out of date[5]. Meaning, they did not upgrade SSH on their server from when it was first setup with the service in 2006. The group Anonymous broke in with relative ease since there were five years of known exploits for the version PSN used. The group leaked about 100 million user data sets with personally identifiable information. Users were rightfully fearful that their data had been stolen and were angry that the network was down during the attack. Sony compensated each user with a temporary 30 access pass to their Play Station Plus network. This user appeasement cost Sony directly and for a smaller company would be tough to do.

This situation was caused entirely by using out of date software. Failing to upgrade software is easy to do but it can end up costing in the end through downtime, leaked internal data, or another vector. It is terrifically easy to set up automatic updates but the cost is hard to ignore.

Facebook moves very quickly in their development process. They even advertise that fact on their jobs page. The average Facebook employee is also relatively young since they try to keep the startup culture despite being one of the popular websites on the Internet. Their programmers are as a result of their age, knowledgeable of many subtleties of working out of college. Facebook is often plagued but simple logic bugs. These bugs can be embarrassing for those affected by them. A couple years ago, Mark Zuckerberg's–the founder–personal photos were publicly accessible and led to both embarrassment and anger from the young billionaire.

2.2 Why Care?

Perhaps the best reason to prevent these sort of bugs from proliferating and evolving into catastrophes is personal pride. It is embarrassing to be the source of a public security bug. One would not want their name on a piece of software that does not work as it is supposed to. A developer should be proud of his or her code. So he or she should take time and effort to ensure a module works properly. Even in a startup, a common goal is to write software that will not have to be rewritten for as long as possible. As a result, the skilled programmer should take pride in how long their software as gone without heavy modification. If the software one haphazardly wrote is found to be the source of a major security hole then the developer responsible should at the least feel embarrassment.

In a startup environment, each person in the company represents a non-trivial percentage of the company. So to write software with a hole in it that necessitates a rewrite is an unproductive use of company time. In the interest of most effectively using time to benefit the startup, each developer should try to write software that will not harm the startup. That said, no developer writes software with the explicit purpose of hurting their employer. However, to prevent a waste of time and money a few action items should be implemented.

3 Action Items

Often times the easiest way to solve a software bug is to explain the impasse to a coworker, listening ear, or teddy bear. Just explaining the problem leads to the solution. The same is true for finding a hidden logically error. Code review does not need to be a formal process in the development cycle but it should be present. It will help mitigate logical flaws like those found on Snapchat, Instagram, or Nomic. Taking 20 minutes to explain if a module sounds efficient, safe, or just ingenious to a coworker can help find small bugs that would not have otherwise been discovered until it was too late.

Auto-updating software has been around for decades but the feature is frequently disabled to ensure that software will not break from an upgrade. Having a beta server on which to test changes before pushing them is the ideal place to test a software stack with updated software. It is just before the users get at the software and it is at the layer where every module comes together–so ensuring everything works together is possible. Updating software on a continuous integration server (i.e., a server that is constantly up to date with the latest software pushes) should not take long and it can save future growing pains.

The biggest change in not just startups but also large companies should be in the hiring process. Many companies and startups ask strictly technical questions–language trivia questions. This line of interviews demonstrates a knowledge of a specific language or framework but does not get at the heart of good, safe development. Programmers need to be able to think about a problem from several different perspectives. Startups should hire developers who think about how a module should work and how it can be broken. A change from the bottom layer–hiring–will have a huge impact on the end product. It would be unreasonable to request a working knowledge of security from every developer but expecting each developer to think a lot about a feature before implementing it will prevent logical bugs from arising from the start.

4 Conclusion

A startup's time must be used extremely effectively with limited resources, compared to larger companies. A startup cannot spend months ensuring their product is bulletproof. Instead all it takes are small changes over time to prevent something catastrophic from happening at all.

From the start, a startup needs the right people. A

startup needs to hire some generalists who can start the frameworks for everything and from there hire specialists who can fine tune the product's different aspects. Even for the fast-moving generalists, they need to be able to architect a sound system that will not be rife with security holes later down because of its initial design. So the first set of developers need to be the best generalists out there not only because they can do everything but also because they can think about how everything should be done. Later down, hiring people that think out problems entirely will only help the company.

Very early on, a startup will stop deploying changes directly to the master server. Quickly, secondary, tertiary and more servers will be established. Throughout the pipeline, peripheral software needs to stay up to date to prevent a PSN-like catastrophe from happening. Setting up auto-updates is easier than trying to fix a server that has been overtaken by hackers. It does not take much time and is a simple trick to keeping attackers out.

As the startup grows and acquires more developers, there become more and more people off whom to bounce ideas. At a certain point it becomes easy to ask one's developer team to take a time out and explain the state of the software stack. It is an approachable forum to ask questions about potential problems that might arise from buggy logic. Catching logic bugs early will save time in the end–the more ingrained logic errors become in the stack the harder it will be to extract them.

There are plenty of other small low-hanging fruit to grab to keep a startup safe. There are more small holes that can cause problems than problems the large one will cause, namely an injection attack. Being mindful of the small problems that can arise and preventing them early is the best defense against the dark arts.

References

[1] Ashlee Vance and Michael Riley, *Tech startups take risks overlooking security.* SF Gate, http://www.sfgate.com/technology/article/Tech-startups-take-risks-overlooking-security-4889205.php, October 11th, 2013.

[2] Bob Nisco, *Snapchat's Caching Snafu.* bobnisco.com/blog/view/snapchats-caching-snafu, December 17th, 2012.

[3] Jason Fitzpatrick, *Firesheep Sniffs Out Facebook and Other User Credentials on Wi-Fi Hotspots.* Lifehacker, lifehacker.com/5672313/sniff-out-user-credentials-at-wi+fi-hotspots-with-firesheep, October 25th, 2010.

[4] Jason Kincaid, *Yet Another Hot Startup Leaves A Gaping Security Hole In Its iPhone App.* TechCrunch, techcrunch.com/2010/11/18/yet-another-hot-startup-leaves-a-gaping-security-hole-in-its-iphone-app, Nov 18th, 2010.

[5] Mohit Kumar, *Anonymous leaks PSN SSH Logs, Sony is responsible for Data Theft?.* TheHackerNews, thehackernews.com/2011/05/anonymous-leaks-psn-ssh-logs-sony-is.html, May 20th, 2011.

[6] Nathan Ingraham, *Lavabit founder closed his secure email service to 'protect the privacy' of its users.* The Verge, theverge.com/2013/8/10/4608664/lavabit-founder-closed-his-secure-email-service-to-protect-the August 10th, 2013.

[7] Quentin Hardy, *Technology Workers Are Young (Really Young).* New York Times, bits.blogs.nytimes.com/2013/07/05/technology-workers-are-young-really-young, July 5th, 2013.