Metasploit and Money

Black Hat DC 2010

hdm[at]metasploit.com

Abstract

In 2008, Metasploit expanded from a community-run project to a corporate product managed by Rapid7. This paper focuses on the transition, the lessons learned during the acquisition process, the challenges of maintaining a community, and the latest improvements to the Metasploit Framework. The points covered are valuable for anyone building an open-source product, contemplating the purchase of one, or considering using an open source product to build a commercial application.

Introduction

Metasploit began as a single-developer tool, grew into a small group of core developers, and then exploded as community-based project before being acquired by Rapid7 and morphing into the company-sponsored, community-driven product that it is today. Throughout this seven-year process, the codebase has changed languages, licenses, and development teams, all the while staying true to the mission of providing useful information and tools for security professionals. This paper outlines some of the challenges, successes, and failures as the project evolved from a crappy video game to a tool that thousands of professionals rely on today.

Origins

In 2003, I was working for a small managed security service and consulting firm; the penetration testing team, of which I was part, depended on a massive stack of open-source and somewhat-underground tools to get the job done. After three different efforts to organize, validate, and sanitize the exploit database, it became obvious that a better solution was required. Just like most consulting shops, we had our own suite of custom scripts and tweaked public tools, but the pace of work was such that we never had time to build anything useful for the long-term. The management team made it absolutely clear that with the workload, there was no time to focus on a development project, and based on the forecast, there would not be time to do so in the future.

Like most folks who are told to ignore what they feel is a important problem; I decided to just do it myself anyways. Since I was doing this for fun anyways, I decided to make a game out of it, the target network would become the "map", the "weapons" were the exploits, and points were scored when the player's specific agent was installed on the target systems.

Over the next few months, I ramped up my ability to write exploits, tweak shellcode, and find new return addresses. After six months of work, I had identified a short list of common problems affecting almost all public exploit code. Public exploits often lack basic information about the constraints of the vulnerability that are required to make small modifications possible without a test environment. Hardcoded return addresses, undocumented space constraints, missing lists of restricted characters, and embedded, obsolete shellcode all contributed to making typical proof of concept exploits useless in

a professional environment. To create future-compatible exploits, a new design was needed, one which evolved into the Metasploit Framework.

When in doubt, do it yourself.

Metasploit 1.0

In October of 2003, version 1.0 of the Metasploit Framework was released. This version was written in Perl, used a ncurses menu-based interface, and included 11 exploits. The feedback from the release was anything but positive; for the folks who didn't just ignore the project, the common response was that it was either a terrible idea, or so poorly implemented that I should just drop the project and move on to something more worthwhile. One person responded with a particularly nasty email, ripping on the code quality, the overall design, and generally saying how awful the whole thing was. This person was spoonm he became the second developer on the project.

The "your code sucks, I can do it better" meme has been a motivating factor since the early days of the project. The project thrives on criticism and one person's complaint can easily result in a better tool for the entire user base. When it comes to community-based projects, you can't afford to have ego - even the most ignorant user can still highlight areas for improvement in the project. This "noob-friendliness" can be trying at times, but can pay off months later when the "noob" becomes either a vocal supporter of the project or a new developer.

Be nice to your critics, be nice to your noobs.

Metasploit 2.0

In April of 2004, version 2.0 was released by myself and spoonm. The code was a complete rewrite from version 1.0, included a brand new module format, 19 exploits, and 27 payloads. At this point, the feedback switched from "this is crap" to "this is a crappy script kiddie tool"; most seasoned exploit developers wrote the release off as some kind of ugly wrapper around other people's exploits; missing the fact that all 19 exploits had been rewritten from scratch, and in most cases, massively improved. The good news is that this release was just solid enough to attract a new developer to the project - Matt Miller, also known as skape. What spoonm did for the Metasploit architecture, skape did for the payloads and encoders. Within two months, Skape had integrated his DLL injection work and written the VNC Injection payload that is still used today.

Over the next two years, the project improved at breakneck speed, attracting dozens of external contributors and a handful of part-time core developers. The dirty secret behind many of the third-party contributed exploits is that one of the core project members rewrote large portions of the module before committing it to the tree. In essence, most new contributors had their first two or three modules completely rewritten before it was accepted into the public repository, with no credit given to the developer who performed the work. As contributors ramped up, newer modules needed much less work, and eventually reached the point where they were committed as written.

Credit is main form of currency in the open-source world; in most projects, nobody is getting paid to work on them part-time, let alone to send in a piecemeal contribution. By spending the time to make sure that an outside contribution is not only included, but matches the standards of the project, you make the project look good, and the contributor look great; which results in even more contributions.

This technique is applied to more than just modules; one of the greatest thing about having an active user base of security researchers is the amount of new ideas. The challenge is finding a way to implement an idea from a contributor, when that person doesn't have the necessary skill set, time, or experience with the project to do the work themselves. Many of the new features added to Metasploit originally arrived as a patch that could in no way be applied to the tree without breaking a dozen other things. The key is to dedicate time to not just implementing these ideas, but also making sure the original contributor knows how to do it next time, all the while preserving their credit for the idea and sometimes even the implementation.

Make your contributors look good, and your project will too.

Perl's Death Spiral

Metasploit was originally written in Perl simply because that was the language I knew best; I could write a little C, a little ASM, and a few other things, but Perl let me implement new tools really quickly with a minimal of hassle. As the Metasploit Framework grew, Perl finally started to show its age, specifically around object oriented programming and Windows platform support. The final straw for me was the Perl 5.8.0 release, which forced all strings to be unicode by default. This change immediately broke all of shellcode and binary protocols in the framework. The lack of real threading support, the instability of ActivePerl for Windows, and the gimpy OO environment forced us to look at new languages for a rewrite.

After considerable debate, where options ranging from C++ to Haskell were suggested, the core development team finally landed on Ruby. The object model was a perfect fit for what we wanted to do with exploits and payloads, the threading support, while not great, was at least usable, and the list of supported platforms seemed to hit all of the common operating systems (including Windows).

A rewrite from one language to another is not for the faint of heart, especially when you have 40,000 lines of especially tough to test code. The process was not simply a port; design changes required every line of code to be rewritten from scratch. The upside is that the new design gave us a massive amount of room to grow. The Ruby-based framework exploited the language's open object model to provide a flexible way to add new features and modifying existing ones. Finally, after eighteen months of development, we had Metasploit 3.0. First, however, we had a licensing problem.

Make the right decisions for your development team, not based on how it might affect contributions.

Boondock Corporation

Around the time that we started to hit our stride during the rewrite from Perl to Ruby, we noticed that an existing security company (lets call them Boondock Corporation), released a commercial exploit tool, written in the same language as Metasploit 2. To make matters worse, the initial list of exploits was almost identical to the list of exploits included in Metasploit 2. Naturally, the development team wasn't happy about this, but since 2.x was provided under the wide open Perl Artistic License, it was perfectly acceptable from a legal standpoint. The issue wasn't that someone derived a commercial product from our open source code, it was that they never contributed back to the project. Later on, it became clear that even though the exploit code was not a line-for-line copy, they were based on the modules from the Metasploit 2 source. Additionally, the shellcode used by this product was only a few bytes different from the shellcode included in Metasploit. The time had come to discuss licensing.

After a series of heated discussions between the development team, we decided on a free-as-in-beer license and hired a lawyer, with the end result being the Metasploit Framework License. To make this license enforceable, a company called Metasploit LLC was created and each of the core developers assigned their copyrights to this company. In return, the company provided each developer with the equivalent of a personal BSD license, allowing for forks, derivatives, and commercialization if any developer so chose. To protect the Metasploit name; a trademark application had been filed. At this point, Metasploit LLC owned the domains, trademarks, and copyrights to the Metasploit Framework. The professional services and filing fees for all of the above totally approximately $3,000 USD, but they were absolutely necessary for us to have a solid legal stance when it came to copyright infringement.

The Metasploit Framework License was used for the initial 3.0 release as well as version 3.1. This license contained an explicit clause about contributions:

```
"You hereby grant and agree to grant a non-exclusive royalty-free
right, to both (i) Developer and (ii) any of Developer's later licensees,
owners, contributors, agents or business partners, to distribute Your
Enhancement(s) with future versions of the Software provided that such
versions remain available under the terms of this License (or any other
later-adopted license(s) of Developer)."
```

The great thing about this license is that we didn't have to force contributors to explicitly transfer copyright to the LLC with each submission. The license terms allowed us to roll changes into the main tree without making it a hassle for contributors. This license went through a few minor changes by the time that 3.1 was released, but was otherwise a great way to avoid another Boondock.

Invest in the legal work, without it the project is vulnerable to unscrupulous asshats.

Metasploit 3.2

In early 2008, both spoonm and skape parted ways with the project to focus on their real jobs. This left myself as the only core developer with a growing mountain of work to do. The solution was again

licensing; I actively recruited the most motivated contributors to take an active role in the project, and in return released the entire Metasploit codebase under the new BSD license. The change from free-as-in-beer to BSD made it clear that anyone could join the project, contribute, and not lose access to their own work if we changed our license again in the future. By the time 3.2 was publicly released in October of 2008, we were back up to 5 core developers and a dozen part-time contributors.

Using a BSD license for commercially-valuable code opens the project for abuse. To pull it off successfully, you need to own your trademarks, own your brand, and be willing to compete with organizations that will actively steal your work. As long as you can innovate and profit from this innovation faster than your competitors, it really doesn't matter who steals your code. This may sound like a rough position to be in, but its not that different from the commercial software world, where features and methods are copied between competing products all the time.

At the end of the day, the only things that set your project apart from anyone else with the same source code are knowledge of the codebase, a recognizable brand, and the community around it. The better known the brand, the more difficult it will become for competitors to simply copy features wholesale from your codebase. The faster you can move your project, the more resources will be spent by the competition keeping up. The closer competitors follow your code, the more control you have over their resources. The only way to prevent competing with your own code is to write new code even faster.

License decisions can leave you racing against your own codebase, but this can be a good thing.

Rapid7

In spring of 2009, a representative from Rapid7 contacted me about possible integration between Rapid's NeXpose Vulnerability Management product and the Metasploit Framework. I liked the idea and we talked about it off and on over a few months. Eventually, we put together a sample implementation and Rapid7 ran it by a select group of customers for feedback. The approach ended up being the wrong one for a number of reasons, but during the process we realized that a closer link between Rapid7 and Metasploit would make sense for both parties. Rapid7 suggested an acquisition of the project, and after a substantial amount of soul-searching and due diligence, I agreed to further discussions.

My biggest concern was keeping the code free for the long-term; this was actually the easy part, Rapid7 had no interest in closing up the project, as the open source model is what kept the project going for six years and any change would only lead to negative results. Rapid7 would immediately benefit from increased visibility and Metasploit would benefit from a dedicated development team. After we sorted out the general outline of what a post-Rapid7 Metasploit would look like, I agreed to meet with the executive team and board members. Eventually, we found middle ground, where both Metasploit and Rapid7 would immediately benefit from the project and meet existing long-term goals faster.

While NDAs protect the details of the transaction, I do have some recommendations for other folks are considering a commercial acquisition.

1. You will need a business plan, this is something that can take weeks to do correctly, and requires expertise that you may not have (market sizing, financial projections, etc). Expect to spend many, many hours working on this with the help of someone who specializes in plan development. Even if you have no plans to commercialize, you need to demonstrate that you have at least thought it through and could realistically execute on the plan.

2. You will need lawyers, accountants, and most likely an investment banker. The folks you find need to be well-versed in mergers and acquisitions and be willing to demonstrate their experience. The best people in these fields can be extremely expensive and expect cash up front in the form of a retainer. Investment bankers will often expect a percent of the final deal as payment for their services. The acquisition of a small company can cost between 50k and 150k for the seller on average. If the deal falls through, you are still on the hook to pay your lawyers and accountants.

3. You will have to "sell" the plan to a group of folks who have no interest in open source or the specifics of your technology. The folks who brought you in can get the ball rolling, but the executive team and board members are the ones who will be signing the deal. You can have the best technology in the world, but if you can't pitch your plan in a way that is easily understood and backed by market facts, you shouldn't be there yet.

4. You will have to answer hard questions; what happens if Company X takes your code and commercializes it? How did you come up with these financial projections? Have you read the market report X and if not, did you take into account Trend Y? Why would someone pay for support for free code? Why wouldn't your users switch to another tool if you followed your plan? Will someone sue us if we acquire you? Where did your source code come from? Who else is a copyright holder? How do you verify each of the commits?

5. You will need to absolutely know your user base. How many people use your software? Where are these people? What companies do they work for? What position are they in? How often is your software used? What are your web stats for the last year? Is your user base growing? Where will it be in two months? How did you arrive at all of the previous metrics?

6. You will need to know what your limits are before you start negotiations. If the deal includes cash, what this number must be. If the deal includes stock, how the stock value is determined. If the deal involves a major investment of new resources to be a success, what those numbers look like. This is the situation where an investment banker earns their keep, but if you don't have one, be prepared to stick by your guns no matter how stressful the situation may be.

7. You will need to know the real market value of what you are selling. This doesn't mean you should demand nothing less, but any information about similar sized projects and how they are valued is critical to setting expectations. For most open-source projects, the only asset is the brand and the community, and communities are notoriously fickle. What would prevent a

competitor from accomplishing the same goals, but without going through the acquisition process? Identify the value and figure out a general idea of what its worth.

8. You will spend a lot of time getting to know the people on the other side of the table. It is important to keep your cool and stay friendly - after all, these are the same folks you will be working with if the deal is successful. Any bad blood or mistrust during negotiations needs to be resolved before the final paperwork is signed.

Once the initial terms are finalized, you still have a long road ahead in terms of how the legal documents are worded, how disputes are handled, and when intermediate agreements expire. This is the point where your legal team gets to work. No matter how long the final documents are, you will need to know them word for word, and understand what those words mean, not just in a general legal sense, but specifically to the jurisdiction where the agreement would be signed. The same sentence can have drastically different consequences in two different states.

In October of 2009, Rapid7 announced the acquisition of Metasploit. The acquisition was a sale of all assets; instead of acquiring Metasploit LLC as a company, Rapid7 simply purchased the property owned by Metasploit LLC, which included domains, trademarks, and copyrights. Metasploit LLC's other business activities (security training) were not directly affected by the acquisition.

Acquisitions are expensive, lengthy, and easy to get screw up. Get professional assistance.

Community Feedback

One of the biggest challenges with moving an open source project under a corporate sponsor is keeping the community alive. Many folks are happy to contribute to a project that is run on a shoe-string and in the spare time of the project managers, but shy away from putting time into what they see as some company's product.

After the acquisition, a couple of the contributors indicated that they were unhappy that the project was now run by a corporation. After a long, long discussion about the reasoning behind this change of heart, it came down to a difference in perception. Even though Metasploit was previously owned by a company (Metasploit LLC), the fact that it had been acquired by a larger corporation and was being actively funded changed how they saw the project. The flip-side, however, is that many casual users became actively involved in the project, contributing code and reporting bugs, because the outlook for the project looked much better for the long-term.

In the end, the impact of the acquisition on the contributor base was actually a net gain in numbers, but many of the existing developers took a more passive role in the day to day development. With that said, before the acquisition, only a handful of developers were responsible for majority of all written code, with most of the development team focused on their own niche areas. Since the acquisition, the increased speed of development has expanded the gap between occasional contributors and the core

project team. The number of casual contributors continues to grow, so in the long run the amount of community involvement should only increase.

Communities may transform during an acquisition, but will stay loyal if treated right.

Metasploit 3.3

After the acquisition, the first release on the schedule was version 3.3. Prior to this, the last major stable version was 3.2, which was released all the way back in October of 2008. The challenge with version 3.3 was it was the first release after becoming a Rapid7 project, but before the first member of development team could start. This required every waking moment of my time over the course of a few weeks to hack out code, test it all, and filter out the bugs.

After the initial 3.3 release, James Lee, Joshua Drake, and Mike Smith joined the team, letting us knock out three quick point releases before the December holidays. The schedule finally started to settle into 2-3 month development cycles, but the rate of improvements going into the development tree has yet to slow down. The difference between before and after the acquisition has been night and day for the project; we accomplished more in three months than we would have done in a full year otherwise, with all of that work going back into the public SVN tree under a BSD license.

My role has not changed much; the hours that used to be consumed by my previous job are now split between active development and handling the various overhead and administrative tasks. I still spend nights, early mornings, and weekends working on the code, and I am as excited as ever about making the project better.

The big challenge is still the obvious one; how do we pay for a team to work on an open-source project full-time, without destroying the value of the project in the process? We have a few ideas and plan to start implementing these in the coming months. Some of these are straightforward while others will require much more planning to execute successfully; either way, the open source code and development model will stay the same.

Working on your hobby as a full-time job has some overhead, but is still an amazing experience.

Summary

So far, the change from a hobbyist project to a company-backed project has gone remarkably well; the key factors have been staying true to the open source roots and keeping in touch with the community. From the second that news of the acquisition spread, a number of folks voiced concerns over the future of the project and the long-term goals of Rapid7. The best way to address these concerns is to prove them wrong, both by continuing to provide what we always have, and by solving the commercial equation in a way that shows respect for the existing community.

Please see our Black Hat DC 2010 presentation for more information on where the project is going and what we intend to accomplish this year.