


An Overview of Public Platform C2's

 kindredsec.wordpress.com/2019/08/12/an-overview-of-public-platform-c2s/

[View all posts by kindredsec](#)

August 12, 2019



As most of us already know, the concept of a Command and Control (C2) framework has been around for ages, given that the ability to persistently communicate with compromised hosts is critical to running any worthwhile campaign. Historically, the general structure of these C2s has followed suit with how most of our networking communications have been built: as a client-server model. The client, which in this case is an infected host, is configured to communicate directly with a server in order to determine any actions it needs to take. In almost every malware analysis report you read, you'll probably see one or more IP addresses or domain names that the piece of malware (which in C2 lingo is referred to as an implant) is configured to "check in" with and obtain "action requests" from. Once these IP addresses or domain names are identified and made public, defenders simply check their network for hosts calling to those IPs, terminate the implants running on those hosts, then block egress/ingress traffic to those IPs or domain names to prevent future issues with the campaign. Then, predictably, the threat actor, now seeing their command servers noted as an Indicator of Compromise (IoC), switches up their infrastructure, updates their implants to call to different command servers, and continue on with their campaign. This scenario is obviously very much simplified, but its a pattern that we see constantly in the perpetual game of cat and mouse between attackers and defenders.

However, how would this scenario change if the domain name being called by the implant is `drive[.]google[.]com`? What about `reddit[.]com`, or `github[.]com`? Would you, as a defender, be able to block traffic going to those domains to prevent the effectiveness of the implant? Of course not; the users on your network would be cut off from major sites that may be critical to their day-to-day operations. When you analyze the traffic destined for these domains, would you be able to determine when traffic is malicious and when its your users simply using the services normally? Maybe, though it certainly won't be easy; at least, not nearly as easy as seeing traffic going to `imac2server[.]cn` and knowing right away that the traffic is bad news.

What I have just described is what we're going to be talking about in this writing. A public platform C2, which I'll be abbreviating to "P2C2," is a Command and Control structure that uses legitimate, innocuous platforms to facilitate server-implant communications. In essence, P2C2s use a reputable service as a proxy, masking the identity of the actual attacking infrastructure and hiding C2 traffic within a sea of innocuous HTTPS traffic that makes up the vast majority of today's network communications.

For the past few months, I have been incrementally working on a project called redViper, which is a C2 Framework that uses Reddit for all of its implant-server communications. While working on the project, I've really enjoyed facing the challenges and reaping the benefits that come with using a P2C2. While redViper is very much a proof of concept and isn't particularly sophisticated (especially the current implant code, which is largely just a placeholder), I nonetheless am really happy with how the project turned out, and it has solidified my belief that using a public platform as a means of C2 communications could actually be a viable solution attackers, especially in environments where traffic is extensively filtered. In this writing, I'm going to discuss the advantages, disadvantages and use cases of P2C2s, using not only my experience building redViper, but also using examples of P2C2s that already have been used in the wild.

Advantages of a P2C2

When it comes to covert channels and communications, I am a strong believer in the concept of obscurity through uniformity. From a defensive perspective, threat hunting and security operations are based almost solely on the concept of anomaly detection. By asking "what traffic and activity aren't normal to my network?", and having the means of finding that traffic, defenders can oftentimes catch malicious activity. By definition, the concept of uniformity nullifies the concept of anomaly. Think of, for example, school uniforms. When students are able to wear whatever clothes they want, every person appears unique and differentiable; even if you couldn't see their faces, you could easily differentiate each student from one another based on the difference in clothing. When students are forced to wear a school uniform, however, it becomes much more difficult. Sure you could maybe identify each individual by their body shape or height, but as you can imagine, it becomes a lot harder to tell who's who.

This is what I find the ideal scenario to be when it comes to C2 communications; if my malicious traffic is wearing the same uniform as Karen-from-accounting's web surfing, defenders are going to have a tough time finding my needle in their haystack of standard web traffic. We can actually see attackers chase this desired uniformity over time in the evolution of C2s. Initially, many C2s used their own binary protocols that used some arbitrary port to communicate on, which quickly became easy to catch. Nowadays, most notable C2s used by threat actors use some form of HTTP or HTTPS for communicating, which makes the task of finding C2 traffic a bit tougher since, as we talked about earlier, HTTP(s) is what most standard network traffic consists of. P2C2s take another step in this direction by saying "you know what, not only am I going to use the same protocol as most of your network traffic, I'm going to go to the same PLACES as most of your traffic." It is this concept that encompasses the first core advantage of P2C2s: the natural innocuity of the communications. Without diving VERY deep into the traffic content itself, the traffic being

sent from and to implants looks completely innocent; Its over an encrypted HTTPS connection, going to a well-known, trusted site. To the average eye, it looks like a normal user using some commonly used website.

What does this mean? Well, first of all, it nullifies essentially every security tool or application that operates lower than Layer 7; packet filtering is useless, stateful firewalls are useless, and even many application firewalls will be of very little use. By using a P2C2, you have made almost every firewall on the planet a waste of metal. Second, it drastically reduces the chances of a blue teamer ever taking the time to analyze your traffic. If I see a mass of traffic going to thnxforthedata[.]ru, a domain I've never seen in my entire life, I'm going to question the legitimacy of that traffic and take a closer look at it. If I see a mass of traffic going to reddit[.]com, my first thought isn't, "well I should probably look into this." I'd assume, as well as almost every person on the planet would as well, that the traffic going to reddit[.]com is, well, someone browsing reddit. The discovery of P2C2 communications is almost always going to be from the discovery of an IoC other than the network traffic itself (discovery of implant, malicious commands being detected, etc), which means that if I build potent implants and operate responsibly, my dwell time will likely exponentially exceed what it would be if my implants were calling back to a suspicious hostname/IP or using a custom protocol.

The second core advantage of P2C2s is the inconsequentiality, and as a result the pseudo-resilience, of an attacker's infrastructure. Think about how you as a regular user traditionally interact with, for example, Reddit: you use the browser of a desktop computer, login to the Reddit website, then you go about using the site. Did it matter what browser you used, or the computer you logged in from? Would you be able to achieve the same results if you were on an entirely different host, in an entirely different network, using an entirely different browser? Of course. In order to use the Reddit platform reliably, the only thing that must remain static and available is Reddit itself. If your desktop computer explodes while you're using Reddit, you don't permanently lose the ability to access your account; you simply just have to use a different computer.

The same is true for an attacker utilizing a P2C2. The actual system that's running the C2 server application is largely inconsequential. All the C2 server is really doing is just logging into a public platform and interacting with it, just like a normal user does. If for some reason an attacker's infrastructure is burned, becomes unusable, or has access revoked, it does not nullify or doom the P2C2; The implants are still talking to the public platform, and will continue to do so even if the system the attacker was working from is dead, since the implants don't have the inclination or even the CONCEPT of attacker-controlled infrastructure. If I, as an attacker, need to move my core operations to another system, I can do so with great ease; I just have to re-run the server application using the same credentials from another host, and from the implants perspective absolutely nothing has changed. This

creates the concept of a notional infrastructure: The attacker is of course operating from somewhere, but since that source can't possibly be extracted from the P2C2 independently, and can change with phenomenal ease, that infrastructure might as well only exist in theory.

Again, this generates two advantages, resilience and anonymity. It builds natural resilience because the functioning of the P2C2 is dependent on the uptime of the public platform, NOT the attacker's infrastructure; and, generally speaking, popular public platforms will be far more resilient and reliable than disparate hosts under the control of a malicious actor. Second, because all the implants communicate directly with a public platform, and not an attacker-controlled server, the location, composition and even the existence of the attacker's infrastructure is unknown to defenders. The most that can be known is what accounts are being used by the actor on the specific public platform, which gives very little information regarding the actor itself. In other words, actors who use P2C2s will no longer be attributed to a set of IPs and domains, they'll instead be attributed to accounts, which are far more disposable, flexible and replicable.

All of this together paints a picture that the utilization of a P2C2 instead of a traditional C2 can make both the detection and attribution of an actor exponentially more difficult in the right circumstances, which is obviously very advantageous to an attacker.

Disadvantages of a P2C2

Just like everything in life, garnering positive outcomes is often going to come at the expense of other things. That remains true for P2C2s as well; what we gain in anonymity and covertness, we lose in control and independence. Let's talk about the problems that could be ran into when using a P2C2 from two different perspectives: a legally contracted red team, and an actual criminal threat actor.

The primary issue of using a P2C2 as a red teamer or pentester is moreso a legal one than a technical one. I am by no means a law expert so take what I say with a grain of salt, but from my understanding it would be quite difficult to use a P2C2 within the context of a contractual agreement with an organization, because in many ways you're taking potentially private information from that organization and placing it on a public platform. If you run the "cat /home/bob/employee_info.txt" command using a P2C2, what you're essentially doing is telling your customer's computer to take private organizational data, and place it on a third party's infrastructure. Now, in the case of redViper, and in most P2C2 implementations, this data will be encrypted and protected in a multitude of ways (In the case of redViper, the data is encrypted and sent on a private subreddit with restrictive access), so the chances of an unauthorized person seeing this data is quite low. However, this doesn't change the fact that the confidential data of one party is being temporarily placed on the infrastructure of another intermediary party which has not been tied into a contractual agreement. Based on my very

limited knowledge of the legal side of penetration testing, this could cause some issues, and prevent P2C2s from being able to be properly tested against corporate networks, which in some ways is another advantage from the perspective of real threat actors.

In terms of an actual threat actor, there exists a few disadvantages as well, both technical and non-technical. I think its important to highlight these disadvantages, because understanding the issues real threat actors may face helps us anticipate and act against these threat actors. First and foremost, in the context of criminal activity, there is an inherent risk involved in utilizing the services of a legitimate platform. In almost every circumstance, legitimate platforms have a means of reaching any and all content that traverses their service. In the case of Reddit, for example, anything posted on a subreddit, private or not, CAN be accessed by Reddit retroactively, even in the case of that post or comment being deleted. So what does this mean for threat actors? Well, it means that law enforcement has an opportunity to access vast amounts of communications done by their C2, which can reveal troves of information about their campaign that can contribute to legal proceedings. If law enforcement is able to obtain a single implant containing credentials related to your P2C2 activity, they could then in theory obtain legal authority to get ALL activity for that specific account from the associated platform. This risk does not exist when using a traditional C2s, since communications are shared between the implant and an independent server that the attacker controls, with no other parties in the middle (barring ISPs). Therefore, a serious criminal actor who has to legitimately worry about law enforcement will always have that Achilles heel scenario where the platform they're using pulls the rug out from under them, which could ultimately land them in a jail cell.

Another issue that may arise, which exists for both legal and illegal actors, is the lack of control the attacker has over their communication infrastructure. When an attacker has full control over the infrastructure running a C2, that attacker can explicitly determine when the infrastructure is available, when it's down for maintenance, and when it needs to "lay low." This same control does not exist if you're using a public platform. The attacker doesn't decide when their communication platform is available; another party does. If Reddit goes down for maintenance, or has some sort of technical issue, attackers using redViper are essentially in the dark if they are relying exclusively on redViper for communications. Now, in practice, it is very uncommon for major platforms like Reddit or Gmail to be completely unavailable, so this issue is, in my view, not too major. So much so, that you may remember that I highlighted the uptime of public platforms as an advantage to P2C2s. However, the lack of control is still something that has to be taken into account.

Now that some of the advantages and disadvantages of these C2s are drawn out, let's touch on two miscellaneous topics that I feel are also important to note.

Viable Platforms for a P2C2

The first miscellany I want to talk about is what makes a public platform viable for use with a P2C2. While nearly any public platform can be manipulated for communications, there are certain features that make a platform more viable for scalable campaigns.

Firstly, in order for a platform to be utilized in a coherent P2C2 the platform must have a somewhat mature and accessible API. Programmatically interacting with any sort of service requires an API that developers can easily utilize; this remains true for P2C2 developers as well. In the case of Reddit, I was able to use *praw*, which is a comprehensible python module that interacts with the quite feature-rich Reddit API. Without this mature API, it would have been much more difficult to build a working framework, likely to the point where developing the P2C2 over the platform would be nonviable.

The next requirement is disposable and easy-to-obtain accounts and API keys. In my view, in order to build a truly secure and formidable P2C2 campaign, an attacker needs a multitude of accounts that can be used to interact with their select platform. This allows implant builds to be dispersed between an abundance of accounts, reducing the damage done of having a single account from your campaign exposed. In the case of Reddit, creating an account requires no confirmed email, no API “justification request,” and no account requirements (there’s one exception to this, but I’ll disregard it for the sake of brevity). While developing redViper, I was able to spin up new Reddit accounts to hook to my implants in less than 5 minutes. Compare this to a platform like Twitter, where you need a confirmed email address and an API justification form filled out and approved, and its easy to see why Reddit is generally more fit for P2C2 activity compared to Twitter.

The final requirement is the ability to control access to your communications within the platform. Currently, there are three platforms in which I have working P2C2 PoCs on: Discord, Google Sheets and Reddit. Each of these platforms serve my purposes because there’s a way to control access to communications: For Discord, an invite only channel can be established so only implant/master accounts can send or receive messages. For Google Sheets, permissions for a sheet can be selectively granted to Google accounts. For Reddit, an invite-only private subreddit can be established in which only implant accounts are invited. In all of these cases, unauthorized access to C2 communications can be prevented by only allowing accounts associated with a campaign to see the communications. The only way a party other than the attacker can see communications is if one of the accounts are compromised.

In my opinion, a platform that meets these three core requirements can very well be a viable solution for building a P2C2 framework. With that, let’s jump into the next topic, where we briefly explore past examples of public platforms being used for C2 purposes.

Real Examples of P2C2 usage

As stated in the beginning of this writing, I have frequently used redViper as a discussion point because its a project that I comfortably understand. However, redViper is far from the first example of a P2C2; there have been multiple instances of public platforms being used by real threat actors. Let's briefly dive into some examples of real threat actors utilizing public platforms for malicious purposes.

1. **LOWBALL:** The LOWBALL backdoor, utilized by APT "admin@338," used the Dropbox service for C2 communications. Command requests are sent as files to the Dropbox platform, which are then downloaded and executed by implants, with output being uploaded back to Dropbox. Implants have a Dropbox Bearer API hard-coded into them, allowing access to the Dropbox account. [Here](#) is a full writeup from FireEye for more details.
2. **ROCKRAT:** The ROCKRAT Trojan was discovered by Cisco's Talos Group, and was found to use Mediafire, Yandex and Twitter Cloud for C2 and data exfiltration activities. The threat actor utilizing this RAT has yet to be confirmed, though its largely assumed to be North Korean. [Here](#) is a full writeup from Cisco Talos for more details.
3. **DarkHydrus:** DarkHydrus is an Iranian threat group that utilizes the RogueRobin trojan. This malware uses primarily DNS tunneling for C2 communications, however newer C# versions of the malware have an "x_mode" option, which makes the implant use the Google Drive API for C2 communications instead of DNS Tunneling. [Here](#) is a full writeup from Palo Alto's Unit 42 group for more details.
4. **Backdoor.Makadocs:** The Backdoor.Makadocs malware is a trojan targeting Windows 8 systems that uses Google Docs for C2 communications. [Here](#) is a full writeup from Symantec for more details.
5. **Turla:** The Turla group, a well known Russian APT, used a malicious browser extension which served as a javascript-based backdoor. Within this extension, C2 information is obtained by parsing comments on Brittany Spears' Instagram posts. This is more of a stager than an actual C2, but I thought it would still be interesting to note. [Here](#) is a full writeup from ESET for more details.
6. **MiniDuke:** The MiniDuke backdoor parses the posts of a malicious Twitter account to obtain a URL for its C2 server. Like Turla's implementation, the MiniDuke malware uses public platforms more for staging than actual C2 communications, however I still wanted to note it. [Here](#) is a full writeup from ESET for more details.

These are just some of the examples of public platforms being used in real campaigns. The fact that legitimate threat actors are continuing to use public platforms as a means of C2 communications is a strong indicator that this threat vector will remain potent for the foreseeable future.

Summary
