BRANDON ENRIGHT, GEOFF VOELKER, STEFAN SAVAGE, CHRIS KANICH, AND KIRILL LEVCHENKO

Storm: when researchers collide

Brandon Enright is a network security analyst at the University of California, San Diego. He is primarily interested in malware and exploit research.

bmenrigh@ucsd.edu

Geoff Voelker is an associate professor of computer science at the University of California, San Diego. He works in computer systems and networking.

voelker@cs.ucsd.edu

Stefan Savage is an associate professor of computer science at the University of California, San Diego. He has a BS in history and reminds his colleagues of this fact any time the technical issues get too complicated.

savage@cs.ucsd.edu

Chris Kanich is a PhD student studying networking systems and security at the University of California, San Diego.

ckanich@cs.ucsd.edu

Kirill Levchenko is a PhD student at the University of California, San Diego. His research is focused on network routing and security.

klevchen@cs.ucsd.edu

WHEN IT COMES TO INTERNET THREATS, few topics get researchers and the media as excited as the propagation speed and vitality of modern malware. One such example is the SQL Slammer worm, which was the first so-called Warhol Worm, a term used to describe worms that get their "15 minutes of fame" by spreading at an exponential rate—infecting every vulnerable machine in under 15 minutes [1]. It is ironic, then, that the latest malware to capture the attention of researchers is not one of the shortest-lived but one of the longest, largest, and most successful bots ever: Storm.

Storm got its name from a particular self-propagation spam email subject line used in early 2007: "230 dead as storm batters Europe." Storm, also known as the Storm worm, is not actually a worm. It is hybrid malware: part worm, part bot (a program designed to perform automated tasks), part Trojan, and even part Peer-to-Peer (P2P) client. Unlike SQL Slammer, where all of the research and analysis is, by necessity, post mortem, the architecture and longevity of Storm have made it a veritable researchers' paradise, offering nearly endless opportunities to measure, interact, infiltrate, and even manipulate it. However, this interactive quality is also a researcher's greatest enemy; the ease with which anyone can poke at the Storm network means that to gather any meaningful research data requires identifying and filtering out the noise created by other researchers attempting to accomplish the same task.

History

The Storm botnet that exists today is not the same network or architecture the authors designed and built originally. In fact, the authors have modified the network architecture on more than one occasion in response to enterprising researchers, making life difficult for them. Our player in this cat-and-mouse game is Stormdrain. We originally wrote Stormdrain in early June 2007 as a way to locate existing and future Storm infections on our institution's network [2].

Originally the Storm bot was little more than a P2P client and downloader; the authors would publish URLs to download new components in the P2P network and new functionality (such as the ability to spam) would be downloaded as stand-alone packages. Within a few months the authors designed and built a new tiered architecture to relay commands and adapted the existing command and control (C&C) P2P network to be an overlay/rendezvous system only. This new network design also allowed the authors to combine all the functionality that was once in separate components into a single unified package that could do everything. By mid-2007 Storm had garnered considerable research and media attention and in October the authors chose to separate the Storm P2P network from the original Overnet/eDonkey network by using a simple encryption scheme. Each big change has slowed researchers down and required additional efforts to continue to track and measure the network. The evolution of the Storm network now makes it necessary for researchers to mention what version and at what time their work on the Storm network took place. Unless otherwise noted, the network architecture discussed in this article is the one that was deployed in May 2008.

A View from Orbit

Much of the research interest in Storm is directly related to its novel architecture. The Storm network is actually two separate networks, each with its own protocol and function. The first is the UDP-based private P2P overlay network built on the Overnet protocol. It is this P2P network that gives Storm much of its resiliency [3]. No actual C&C between the Storm authors and the bots uses this P2P overlay network. The network's only function now is to facilitate advertising and locating proxy addresses in the separate TCP C&C network. This second protocol and network utilizes a custom TCP- and HTTP-based protocol for C&C. This C&C network makes heavy use of proxying to provide several layers of anonymity between the authors in control of the network and the actual bots doing their dirty work [4]. The Storm authors run a series of proxies on the TCP C&C network and UDP P2P network we call controllers. These controllers provide HTTP proxy services on the TCP network and locate Storm infections on the UDP P2P network to turn into proxies. Even though no actual bot C&C uses the P2P portion of Storm, all Storm infections participate in the P2P network so they can advertise/locate proxy services to connect to the TCP C&C network. Overnet does not provide any sort of authentication; in true egalitarian fashion, all peers have equal roles. Because of this, Storm's P2P network is an attractive place for researchers to measure and manipulate it. Additionally, any attack or disruption in the P2P network will directly affect both Storm's proper functioning and the work of other researchers. It is difficult to strike a balance between being a good citizen in the network and potentially damaging it through novel research techniques.

The inherently malicious nature of the Storm network engenders a carte blanche, "the gloves are off" attitude in some researchers. Initially Stormdrain was not prepared to handle the constant stream of misinformation thrown at it. For much of its early life, for example, Stormdrain crashed trying to handle various buggy and maliciously crafted addresses (multicast, 127.0.0.1, 0.0.0.0, etc). It wasn't until Stormdrain stopped making assumptions about the protocol, message fields, addresses, and content of the network that it was able to run for more than a few minutes without crashing.

DHT Modus Operandi

To understand how different research groups can affect Storm or each other, it is important to understand how Storm's P2P network works. The network is based on the Overnet Distributed Hash Table (DHT), which in turn is based on the Kademlia DHT algorithm [5]. These names and acronyms are not particularly important. What is important is that DHT networks provide two basic functions: storage of some set of values and, later, retrieval of some of those values. In Storm's case, the values being stored and retrieved in the network are TCP C&C proxy IP addresses and ports. The mechanics of the storage and retrieval of data are at the heart of what so many researchers are poking at. To fully understand how Storm's DHT works, a simple analogy is in order.

Suppose you want to build a human-based social message storage system. One simple method would be to designate one person as the "note holding" person. Anyone who wants to leave a message for someone else can simply go to the note holder and give the holder a note. Anyone seeking that note can go to the note holder to get a copy. This system works pretty well in small groups but isn't resilient—if the note holder gets hit by a bus all is lost. A somewhat related issue to resiliency is that not all participants have the same role or responsibilities. A simple fix to this system would be to make everyone a potential note holder. How could that be accomplished? One solution is to introduce envelopes to put the notes in and create a simple addressing system for the people and envelopes. In this new system notes are not handed directly to a note holder. They are first placed in a numbered envelope and the envelope is given to the person whose own number is closest to the number on the envelope. To retrieve a note, you must find the person whose number is closest to the number on the envelope you are looking for. Without more structure, this process would get quite chaotic.

For the sake of simplicity in our note storage system, the numbers chosen by people or written on envelopes will be limited to 1 through 100. Rather than try to assign people numbers as they enter the system, they will be allowed to randomly choose a number for themselves. Also, to assist in finding numbers, any participant can be asked what their number is. Participants can also be asked to provide the number of any other participant that they know about. To aid in finding numbers quickly, participants will keep themselves in numerical order by standing in a line. With this additional structure in place, a dozen people could easily make a working note storage system. If one of the participants wants to store a note in the system, that person can write out his or her note, put it into an envelope, and number it, then seek others in line whose numbers are close to the number on the envelope the participant wants to store. To accomplish this the participant can approach someone else in the line and ask for his or her number. If the numbers aren't the same, the participant can ask for a list of people that the other knows about whose number is closer to the desired number. The person trying to store the envelope can keep doing this, slowly getting closer to a person with the same or a closer number. The would-be envelope storer keeps doing this until he or she is satisfied that anyone closer to the desired number can't be located, at which time the participant hands a copy of the envelope and note to that person. To reduce the negative effect of people holding envelopes who leave the line, it is best to give a copy of the envelope and note to several close participants. If new participants enter this system, they can choose a random number for themselves and go about using this same search procedure to locate the right place in the line to stand.

With this simple system several basic tasks are easily accomplished. For example, to store a note inside envelope "34" in this line, one could walk up to another and say, "What is your number and all the people you know about close to the number 34?" The response may be something like "I'm number 71 and I know persons with number 56 and 51 that are closer to 34."

The next step would be to approach numbers 56 and 51, asking the same question. The responses may provide numbers 37 and 30. When both 37 and 30 are asked the same question, no closer numbers are learned about. This means that 30 and 37 must be the closest numbers to 34. A copy of the envelope and note can be made and given to both people to hold. Putting new envelopes with notes into the line like this is called "publishing." If someone else comes along and wants to find a previously published note, the same system can be used. To search for envelope 34 the search would most likely end up slightly different. Perhaps the person doing the searching knows about some other set of people. The person can walk up to one of these and ask, "What is your number and all those you know about close to the number 34?" The response could be something like "I'm number 20 and I know about persons 29 and 30." When person 29 is asked, that person may respond with persons 30 and 37. The responses from 30 and 37 will be slightly unique, though. Because each has an envelope labeled with number 34 from a previous publish, they will both respond with others close to 34 and include themselves in the list. If, in the course of searching for an envelope, the person asked responds with him- or herself as an answer, it is an indication that the person has an envelope labeled with the desired number and can be asked for it.

Search failure can also be easily detected. If while searching for an envelope number no new close people are located and none offer themselves as a source then it can be assumed that the envelope does not exist and searching can be stopped. Without people standing in an ordered line the bound on the amount of searching someone would have to do before locating an envelope or giving up would be much higher. Finally, the person doing the searching does not need to start right where he or she is standing; the person can "skip" to the closest person he or she already knows about that happens to be close to the desired envelope. If someone searching for envelope 90 has already had contact with person 95, then that person can start the search with person 95.

Although this "line of people" analogy follows just a few simple rules, it should be readily apparent how dynamic the line can be. With people coming and going, reliability is not guaranteed. To try to keep content (notes inside of envelopes) fresh, the content is published not once but constantly, at regular intervals. Also, a person in the line may know about his or her neighbors at one point but, with people entering and leaving the line often, the line members must constantly search for people close to them and announce their presence to others. Acquainting yourself with another peer is called "connecting" in Overnet. Announcing your continued presence to someone you've already connected to is called "publicizing." Searching for another number (for whatever reason) is called "searching." As discussed before, storing content in the network is called "publishing."

The aforementioned setup is nearly exactly the way the Storm P2P network operates. The primary difference in the systems is in the scale. In the Storm Overnet network, rather than only use the numbers 1 through 100 for people and envelopes, Storm uses 0 through ($2^{128} - 1$), an astronomical number. The number picked by a Storm peer (person) is typically called an Overnet ID or simply OID. The number on the outside of an envelope is typically called a hash or key. Additionally, rather than a mere dozen people in line, Storm has thousands of peers online at a time. The ability to skip around in line to the closest known person is roughly equivalent to the k-bucket system in Kademlia or Chord's finger tables.

Storm's Innovation

The two attributes that have not yet been well defined about this system are how to decide what numbers to assign envelopes and what useful information to actually write down for the notes. One of Storm's most interesting innovations is how it repurposes these pieces of Overnet. Storm has two different types of notes that it wants to publish. The first is used to advertise the ability to become a C&C proxy to the Storm authors. If a Storm node comes online and determines that it is publicly accessible to other Internet hosts, it first tries to announce itself as "proxy-capable" to the controllers.

The other type of note is used by peers who have already been transformed into a proxy by a Storm controller and want to advertise their proxy service to others by generating proxy-advertisement notes. Regular Storm peers that are not publicly accessible and simply want to perform nefarious tasks will seek out a proxy by searching for proxy-advertisement notes. Two different envelope numbering schemes are used but both are based on the current date and each can generate 32 different envelope numbers per day. The first scheme is only used for proxy-capable notes and a hash of the date is used directly. Only the Storm authors search for envelopes with these numbers because only they turn proxy-capable peers into actual proxies (by way of an RSA encrypted packet we term the "Breath of Life" or BoL for short). The second set of 32 envelope numbers is generated from a hash of the date where 1900 has been subtracted from the year. That is, the year 2008 uses 108 as the year. Storm nodes that have become proxies use these envelope numbers to publish their proxy-advertisement messages. Regular Storm peers search out these envelope numbers to locate the proxy service. By keeping the two groups of envelope numbers separate, Storm does not mix up peers looking to be proxies with peers that have already become proxies.

Of course, the trouble with rules is that they can easily be broken. Overnet specifies a simple set of rules and all (nonbuggy) versions of Storm implement those rules. Researchers, however, typically do not. Let's return to our analogy for a moment. If you want to determine every participant in the line (crawl the network) you can pick a random envelope number and search for it. As you learn about new people you can add them to a master list. If you do this fast enough and for long enough your master list will be nearly complete. Many researchers are crawling the network constantly, which can put a significant load on Storm bots and other researchers.

A View from the Stormdrain

One of the benefits of crawling the Storm network for many consecutive months is that anomalies are easy to spot. Stormdrain is set up to graph in real time many attributes gathered while crawling. Sudden bumps or dips are often the result of one or more misbehaving nodes. Initially there was no perfect way to tell which peers in the network were buggy nodes versus impostors under the control of third parties.

Only the most heinous abuses of the network were clearly the doing of third parties. Attempts to classify individual peers based on their behavior were often lost in a sea of data. For example, in late 2007 Stormdrain suffered a sudden drop in efficiency coupled with a large spike in CPU usage. The code was carefully reviewed for possible bugs but none were found that could cause the symptoms. Closer examination of network traffic revealed several IPs in China flooding Stormdrain with Overnet traffic to the point of Denial of Service (DoS). When the IPs were blocked a few hours later the attack resumed from a new set of Chinese IPs. A change had to be made to Stormdrain's `select()` loop to reduce the effect of the attacks. In the line analogy, Stormdrain is just a person walking around and contending with others doing the same. We put a lot of work into developing heuristics to differentiate Storm peers from third parties but none had the 100% accuracy we really wanted.

Through some reverse-engineering work on the Storm binary, it was discovered that Storm had a badly flawed pseudo random number generator that caused the OIDs it picked randomly to follow a specific pattern [2]. Using this knowledge as an oracle allowed Stormdrain to differentiate real Storm peers from impostors in the network. Suddenly it became very easy to measure the effect each of the other researchers had in the network and even to track the various attacks as researchers would evolve and adapt new techniques. It was an epiphany. Using this oracle to separate impostors from Storm has allowed us to create a virtual "police lineup" of others participating in the Storm network that are not Storm bots. Here are a few of the different types of "people" in that lineup:

Babbling Idiots

These people constantly walk up to Stormdrain and say something—anything. Sometimes it's a search; other times it's a publicize. It's never useful, but they just won't leave Stormdrain alone. Stormdrain currently does not bother to blacklist these peers because they aren't harmful enough.

The Deaf

These people periodically ask Stormdrain questions but don't even acknowledge the answers. When Stormdrain queries them, they don't respond. NAT can cause this sort of behavior but NAT has other qualities to it that these people don't exhibit. This also wastes Stormdrain's time but the design of Stormdrain keeps these peers from causing any real harm.

The Schizophrenic

These people answer all queries with imaginary friends. Stormdrain does not know this at first but later queries always fail to contact the peers they claimed to know. This is a very common attack in the network, and Stormdrain uses heuristics to identify and blacklist peers poisoning the network in this manner.

Pathological Liars

Some peers in the network respond with exceptionally poor results. When Stormdrain queries peers like this, the search result is intentionally crafted in a way to throw Stormdrain as far from the right peers as possible. In the envelope analogy, this would be like maliciously responding to a search for number 34 by claiming that person 90 is actually person 32. These peers can reduce the search efficiency of the Storm network but don't negatively impact Stormdrain, because crawling the network does not require actually locating content, just other peers.

Junk-mailers

A person can only hold a small number of envelopes without dropping some. To prevent legitimate notes being found, junk-mailing peers constantly publish random notes to everyone holding a particular envelope number. Those searching for that envelope number will then likely receive only the random notes. In the Storm network this attack is performed against any peer with an OID that happens to be close to the 32 daily rendezvous hash locations. This is known as the Eclipse attack and has proven very successful in tests [6].

Gangs of Friends

These groups of peers all choose OIDs very close to some content hash location. When Stormdrain searches these peers for content, they always respond claiming another in their group is closer. In this manner, these peers will pass any searching node around in circles until the peer gives up, thinking that no content has been published at that hash. This is referred to as the Sybil attack and is used by some researchers to attack the 32 daily Storm proxy-advertisement hashes [6].

Tonya Harding Wannabees

Sometimes Stormdrain gets punished (via a DoS) for searching for particular content. When more than one peer is involved in this behavior the result is a DDoS. Stormdrain has had to contend with attacks like this on more than one occasion. It isn't clear whether this bat-to-the-kneecaps approach is the result of vigilante researchers, rival spam gangs, or some other players.

Researchers Collide

The trouble with having so many easily performed attacks is that invariably they are used often. Indeed, at one point or another all of these attacks have been performed in the Storm network—often several of them concurrently. To successfully crawl the network a researcher must put extensive engineering time into detecting and reducing the effectiveness of each of these attacks. This, in turn, encourages other researchers to perform more stealthy and difficult to detect attacks. There was a joke at a recent security conference [7] that eventually the Storm network would shrink to a handful of real bots and there would still be an army of rabid researchers fighting with each other to measure whatever was left!

Although this is certainly an exaggeration, it leads to the moral of the story: Engineering time and care must be taken not to inadvertently measure the activities of other researchers and network disruptors. Addressing this problem is perplexing because there is nothing fundamental that allows one to make this determination; a sufficiently sophisticated disruptor can design software that is indistinguishable from a bot at the network layer or simply infect a large number of honeypots with the bot itself and manipulate their behavior.

Thus, the question becomes, "At what point is the signal-to-noise ratio so low that there is no meaningful signal left to do research on?" We do not have an answer. One thing is certain, though: The unique architecture of Storm has given researchers around the world an unprecedented view of the future of botnets.

ACKNOWLEDGMENTS

Our thanks are owed to Vern Paxson and Christian Kreibich for detailed discussions and feedback on investigating the Storm botnet, Joe Stewart of SecureWorks for offering his insight into the workings of Storm, and Gabriel Lawrence and Jim Madden for supporting this activity on UCSD's systems and networks.

REFERENCES

[1] The Spread of the Sapphire/Slammer Worm: http://www.caida.org/publications/papers/2003/sapphire/sapphire.html.

[2] “The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff,” *USENIX LEET ’08*: http://www.usenix.org/events/leet08/tech/full_papers/kanich/kanich.pdf.

[3] Storm: http://www.usenix.org/publications/login/2007-12/pdfs/stover.pdf.

[4] “On the Spam Campaign Trail,” *USENIX LEET ’08*: http://www.usenix.org/events/leet08/tech/full_papers/kreibich/kreibich.pdf.

[5] Kademlia: A Peer-to-Peer Information System Based on the XOR Metric: http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf.

[6] “Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm,” *USENIX LEET ’08*: http://www.usenix.org/events/leet08/tech/full_papers/holz/holz.pdf.

[7] First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET ’08): http://www.usenix.org/events/leet08/.