CS 161 Spring 2010 — Computer Security Paxson/Wagner — Discussion 10

1. Botnet command and control

(a) Consider a botnet that uses HTTP for its command and control channel. A bot contacts `http://foobar.com` to get the latest commands from the botmaster. How could law enforcement take over this botnet?

Answer: Change the DNS records. Make `http://foobar.com` point to a server controlled by law enforcement instead of a server controlled by the botmaster. Law enforcement can convince the nameserver `.com` to direct requests for `foobar.com` to a law enforcement-owned site.
Note however that this tactic can become much more complex when it requires international coordination.

(b) The botmaster switches to HTTPS to prevent the above attack from law enforcement. Now when a bot makes a request over TLS, it checks that the server responds with a certificate signed by the botmaster's own CA. Will this protect the botnet from law enforcement?

Answer: Law enforcement can take over the domain, but won't be able to produce a legitimate certificate signed by the botmaster's CA. Law enforcement could however shut down `foobar.com` instead of taking it over.

(c) To further defend against law enforcement, the botmaster changes the bot code so that instead of hard-coding in `foobar.com`, the bot has dozens of domains hard-coded in. The bot will try a bunch of names in the list until it finds one that is registered and has a certificate signed by the botmaster. Now what can law enforcement do to take down the botnet?

Answer: If the names are predictable, law enforcement can try to take down all the names on the list. However, law enforcement needs to take down every domain on the list, whereas the botnet only needs one registered domain to continue functioning.

(d) How can you improve the above scheme to make it even more resiliant to attack by law enforcement?

Answer: Use some trend-based naming scheme. For example, the domain name could be derived as some function of the most popular feeds on twitter. This way the bots can compute a likely domain name and the domain names are constantly in flux, making it more difficult for law enforcement to shut them down.
Another approach is to generate an enormous number of names and have the bots try a random subset, making it very difficult for law enforcement to disrupt all of them.

(e) Can you think of a scheme whereby the botmaster can push out updates and commands to a very large botnet *without* using DNS, and that works without the bots having *any* information about the location of other bots or elements of the C&C infrastructure? You can assume that the bots have wired into them the public key for the botmaster's CA.

Answer: Bots randomly scan the Internet looking for other bots belonging to the botnet. When they find one, each bot exchanges a version number reflecting the latest code update and command that the bot received. The bot with the lower version number then asks the bot with the higher version number for the updates or commands it has since received. *Providing that each of these updates/commands is correctly signed by the botmaster's private key*, the lower-version-number bot accepts them as genuine.

This scheme might seem far-fetched but in fact it's what the contemporary *Conficker* botnet (also a worm, for how it propagates) uses. Indeed, it used all of the more sophisticated schemes listed in this problem other than the trend-based naming scheme. An early version would try a set of 250 random domains that changed every day. Later, the botmaster upgraded the code to try 500 random domains chosen from 50,000 possible domains, as well as adding the *peer-to-peer*-style of searching for updates/commands sketched above.

(f) Discuss the pros and cons (from the botmaster's point of view) of each of the following botnet topologies:

- Star
- Hierarchical
- Peer-to-peer

Answer

- Star. Pros: Botmaster has direct and immediate control of every bot. Control node can ensure that every bot is in sync. Difficult for opponents to infiltrate. Cons: Single node is bottleneck for delivery of new commands. Single point of failure; if communication channel from control node is disrupted or taken over, the entire botnet is lost.
- Hierarchical. Pros: Less of a bottleneck, new commands can propagate more quickly. Cons: If one node high in the hierarchy goes down, all nodes beneath it are disconnected from the botnet.
- Peer-to-peer. Pros: New commands can propagate very quickly. No single point of failure. Cons: To ensure connectivity, there may need to be many redundant connections. Difficult to diagnose problems.

2. Worm distribution

(a) Recall that the Slammer worm spread quickly. Really quickly. What are some technical ways to build a worm that spreads blazingly fast?

(b) Why do we care if a worm spreads quickly? What ever happened to "slow and steady wins the race"?

(c) Suppose you want to capture one of these worms but you do not want to search among many disparate machines in the hopes of finding the worm. How might you systematically try to capture the worm?

(d) What are some of the major concerns/problems with the approach in the last question?

Answer:

(a) i. Localized scanning

ii. Avoid redundant scanning by arranging to split up the work of scanning among the exploited nodes in a coordinated fashion. There's a clever way of doing this known as a "Warhol worm" (for the notion that the worm can get its "15 minutes of fame" because that's how quickly it spreads throughout the Internet).

iii. Precompute vulnerable hosts (a "hit list"). Maybe over a long period of time you build up a list of IP addresses that are vulnerable, or maybe you exploit only a few hosts and have them discover vulnerable hosts. Then, when you have a large enough base, you exploit all of the precomputed hosts at once.

iv. Make your packets as small as possible so more exploits can be sent more quickly, but this often isn't feasible of the exploit is at all complicated.

v. Better IP partitioning that involves splitting the work among the exploited nodes.

vi. Metaservers to find online, vulnerable hosts. For example, use Google's cache to give you IPs of real hosts.

vii. Contagion approach, utilizing the natural contacts of a host, such as peer-to-peer networks, to find viable hosts to attack. This encourages spreading across peer-to-peer networks and other highly connected networks quickly.

(b) i. If you're really slow, then everyone will be patched. However, that would have to be one slow worm.

ii. Outrunning defenses. If your worm spreads quickly, humans, such as system administrators, will not be able to react quickly enough to stop the spread. Thus, systems will need to rely on automated defenses. These are difficult to trust with making decisions regarding disrupting traffic.

(c) You could set up a *honeypot*: a system or set of systems that have no operational purpose and instead are meant to be observed to see who meddles with them. For capturing worms, one technique is to deploy popular systems that are unpatched, so that they are more likely to be exploited.

Defenders then monitor such systems for any unusual behavior and thus determine whether they have been exploited. This is much easier to do than for a regular system because the honeypots are not production systems.

(d) While this is an important tool, there are a number of significant issues to consider.

i. *Escape*. As a honeypot, you are setting up a victim. This potentially means you are knowingly giving the attacker more attack surface and computational ability. Besides being of questionable legality, it generally isn't a nice thing to help spread worms. You might try to confine the worm by using a VM and/or modifying its network traffic; however, see the next item.

ii. *Detection*. You must be concerned about malware detecting that it is within a honeypot. This is of particular concern if you are running the honeypot within a VM, but it is also a problem if you are just modifying the network traffic. If the malware or worm detects that it is within a honeypot, it might shutdown or behave differently.

3. Ransomware

(a) In lecture, Professor Paxson mentioned viruses that encrypt a user's documents, delete the originals, then demand payment in order to have the documents decrypted. Describe how either symmetric or asymmetric encryption could be used for this purpose.

(b) Recently, an intriguing virus called Kenzero has been in the news. Kenzero searches a victim's web browsing history / cache for pornographic websites, takes a screenshot of what it finds, and posts it on a public website with the victim's name. The virus then directs the user to a site where they must make a credit card payment of $16-400 in order to remove the post before Google crawls it.

What other types of threats (apart from deleting or encrypting data) can you imagine a virus making to extract payment?

(c) Instead of making threats, can you imagine any ways viruses or worms might offer incentives or rewards in exchange for payments? Can you think of anything a virus or worm might ask the user to do other than make a payment?

Answer:

(a) Using symmetric encryption, the virus would have to first encrypt the files, then send the key elsewhere (and delete it locally). The remote party would return the key if payment is made. This works so long as the victim's system doesn't record the key before it is deleted.

If asymmetric encryption is used, it would also be possible to generate a key pair ahead of time and embed the public key in the virus while retaining the private key remotely. In this case the user would not be able to decrypt the files even if they record all the keys present on their system (at any point).

(b) Perhaps the most interesting possibilities are those that leverage social connections in some way. For example, the virus could find the victim's email contacts or facebook friends, then threaten to send them something embarrassing. Things that could be sent include porn the victim has viewed, emails or messages to and from other users (especially those that mention the contact they will be sent to), etc. Apart from forwarding existing information, the virus might attempt to make it appear that the victim has taken an action himself (e.g., forge a message, change facebook relationship status). Other conceivable threats include publicly revealing passwords, secret keys, or credit card information and consuming resources somehow (e.g., telling other virus instances to constantly send SMS messages to the user's phone).

In order to ensure the user cannot prevent these threats by simply shutting down their machine, the virus could first set up the process so it will be automatically executed by a third-party instance of the virus unless the user makes a payment within some time limit. Once this is in place it would demand payment.

(c) Again, you can think of interesting incentives based on a user's social connections. The virus / worm could offer to allow the user to read emails or messages from the account of a friend that has been compromised (perhaps showing promising excerpts as a teaser) or allowing the user to watch their friend's webcam / microphone. Similarly, the virus may offer the user the ability to control other infected machines.

More speculatively, one might imagine a virus or worm that somehow asks a user for assistance in propagation (perhaps in exchange for a portion of the resulting payments, in the style of an Amway-type scheme). While I wouldn't expect any users to be willing to sell out their own friends by, say, sending them an email attachment with a virus, you might be able to get them to do it to strangers. It's usually easy to identify accounts on, say, MySpace that are spreading a virus because they are either (a) a phony account that obviously was created by virus and / or (b) making a series of identical wall posts. If the virus is instead spread by a large number of independently motivated, real people, it could be harder to identify.