Paxson
Spring 2017

CS 161
Computer Security

Discussion ∞

Additional Pseudo-Section Material

This material is from past CS161 sections and is intended to aid in studying the final material presented in the Spring 2017 offering which was not covered by section, homeworks, or projects. You are *not* responsible for any additional specifics present in these materials but not present in the corresponding lecture materials.

Question 1 ***Worm Spread*** **(10 min)**

(a) In class we have seen that typical network worms propagate using scanning. Can you think of other ways to spread a worm?

> **Solution:** This question is quite open-ended and has multiple solutions. One solution is to post links to friends via email/Twitter/Facebook. Another is to read logs or other files on the local machine to find other likely victims. A third is to use a search engine ("outsourcing" the scanning). A fourth is to contact a server whose job it is to track other servers (for example, some online games have "meta-servers" that you can contact to find a list of servers available for playing the game).

(b) The typical virus exploits a benign application to execute its own (malicious) code. Exploiting real world applications is getting tougher every year because of the mitigations for buffer overflows that we discussed. Can you think of a way that a virus would not require an exploit to achieve code execution?

> **Solution:** The virus could fool a user to download "critical updates" through social engineering. The *Koobface* worm spread using this technique. Once a user's computer is infected with Koobface, it would post a link on the user's friends' walls. When clicked, the linked page would ask the user to download and install an "update" for their Adobe Flash Player. If the "update" is installed, the computer is now infected with Koobface.

Question 2 ***Botnet C&C*** **(10 min)**

Consider the use of Twitter for botnet command-and-control. Assume a simplified version of Twitter that works as follows: (1) users register accounts, which requires solving a CAPTCHA; (2) once registered, users can post (many) short messages, termed *tweets*; (3) user A can *follow* user B so that A receives copies of B's tweets; (4) user B can tell when user A has decided to follow user B; (5) from the Twitter home page, anyone can view a small random sample (0.1%) of recent tweets.

(a) Sketch how a botmaster could structure a botnet to make use of Twitter for C&C. Be clear in what actions the different parties (individual bots, botmaster) take. Assume that there is no worry of defensive countermeasures.

Solution: A simple approach is that the botmaster registers two Twitter accounts, A and B. (This requires only solving two CAPTCHAs, trivial to do by hand.) A is used to send commands, and B for receiving commands. The bot malware includes within it the credentials for the B account. New bots then access the B account to read the tweets sent by the A account, which encode the instructions to the bots.

An alternative approach would be to create a new account per bot. Doing so requires some method for solving all of the required CAPTCHAs; for example, by purchasing solutions from a solving service available from the underground economy.

A third approach would be for bots to directly use the account of any user on an infected machine who already has an account.

A fourth approach would be for the botmaster to register a single account and use it to generate 1000s of identical tweets for each command they want to send to their bots. Each bot visits the Twitter home page and examines the random sample of tweets there to look for instructions.

(b) Briefly describe a method that Twitter could use to detect botnets using this C&C scheme.

Solution: For the first scheme, Twitter could look for access to the same account from many different IP addresses.

For the second scheme, Twitter could look for accounts whose followers all only follow that account.

For the third scheme, Twitter could filter out tweets from their random sample if the tweet is identical to a previous tweet.

(c) How well will this detection method for Twitter method work?

Solution: The first scheme likely works well—it would be unusual for a legitimate account to be accessed from 1,000s of different IP addresses.

The second scheme might run into trouble for certain types of users, say celebrities, who have many followers who only use Twitter to follow the celebrity.

The third scheme should work well—there's not much value in having an identical tweet show up in the random sample that's made public.

(d) Briefly discuss a revised design that the botmaster could employ to resist this detection by Twitter.

Solution: For the first defense, the botmaster will need to shift to using one of the other two approaches given in part (a) above.

For the second defense, the botmaster could have the bots follow some other randomly selected users in order to look more normal.

For the third defense, the botmaster could add some minor variation to their repeated tweets so that Twitter doesn't view them as identical.

Question 3 ***Botnet Command and Control*** **(7 min)**

(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?

Solution: 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. For example, if instead of `http://foobar.com` used `http://foobar.ru` then it would be difficult for authorities in the US to take over. Of course, the Botnet would have to be smart enough to hide from Russian authorities. Viruses that refuse to do anything if the default language is Russian have been known to exist.

(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?

Solution: 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?

Solution:

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 resilient to attack by law enforcement?

Solution:

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.

Solution:

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

Solution:

- 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.

Question 4 ***Another Use for Hash Functions*** **(15 min)**

The traditional Unix system for password authentication works more or less like the following. When a user u initially chooses a password p, a random string s (referred to as the "salt") is selected (and kept secret) and the value $r = H(p \mid\mid s)$ is computed, where H is a cryptographic hash function. The tuple (u, r, s) is then added to the file `/etc/passwd`. When some user later attempts to log in by typing a username u' and password p', the system looks for a matching entry (u', r', s) in `/etc/passwd` and checks that $H(p' \mid\mid s) = r'$.

(a) In this system, what do you suppose the purpose of the hash function H is? Why not just store (u, p) directly in `/etc/passwd` without computing any hashes? Is there an advantage in terms of security?

Solution: The purpose is to prevent someone who can read `/etc/passwd` from discovering all the user passwords, while still allowing a typed password to be checked against that file.

Even if an attacker must obtain privileges to read the passwords (and therefore could directly access user accounts, or change their passwords), there's still a security benefit, which is that the users may have chosen the same password on other systems. Preventing recovery of the passwords from the password file will then protect those other systems.

(b) What do you suppose the purpose of the "salt" s is? Why not just compute $r = H(p)$ and store (u, r) in `/etc/passwd`?

Solution: Under the described scheme, the best way for an attacker to find a password based on the hash is to try hashing guesses one after another (a dictionary attack). If no salt was included, this could be done more efficiently across many systems by building a big, static database of hashed candidate passwords and checking the contents of various `/etc/passwd` files against it. With salt, an attacker is forced to try hashing each password guess all over again for each account they want to crack. Salt also prevents `/etc/passwd` files from revealing when users choose the same password on multiple systems.

Question 5 ***El Gamal and Chosen Ciphertext Attacks*** **(9 min)**

The lecture notes explain El Gamal encryption as follows. The public parameters are a large prime p and an integer g such that $1 < g < p - 1$; these values are known to everyone. To generate a key, Bob chooses a random value b (satisfying $0 \leq b \leq p-2$) and computes $B = g^b \bmod p$. Bob's public key is B, and his private key is b. If Alice has a message m (in the range $1 \ldots p-1$) for Bob that she wants to encrypt, she picks a random value r (in the range $0 \ldots p-2$) and forms the ciphertext $(g^r \bmod p, m \cdot B^r \bmod p)$. To decrypt a ciphertext (R, S), Bob computes $R^{-b} \cdot S \bmod p = m$.

(a) Suppose you intercept two ciphertexts (R_1, S_1) and (R_2, S_2) that Alice has encrypted for Bob. Assume they are encryptions of some unknown messages m_1 and m_2, and that you have Bob's public key (but not his private key). Show how you can construct a ciphertext which is a valid El Gamal encryption of the message $m_1 \cdot m_2 \bmod p$.

Solution: The ciphertext may be constructed as follows, where all computations are done modulo p.

We have that $R_1 = g^{r_1}$, $R_2 = g^{r_2}$, $S_1 = m_1 \cdot B^{r_1}$, and $S_2 = m_2 \cdot B^{r_2}$ for some r_1, r_2. Define $r_3 = r_1 + r_2$ and compute the following:

$$R_3 = R_1 \cdot R_2 = g^{r_1+r_2} = g^{r_3}$$
$$S_3 = S_1 \cdot S_2 = m_1 \cdot m_2 \cdot B^{r_1+r_2} = m_1 \cdot m_2 \cdot B^{r_3}$$

So (R_3, S_3) is a valid encryption of $m_1 \cdot m_2$. In technical terms, El Gamal is *homomorphic* under multiplication, i.e.,

$$E(m_1) \cdot E(m_2) = (g^{r_1}, B^{r_1})(g^{r_2}, B^{r_2}) = (g^{r_1+r_2}, B^{r_1+r_2}) = E(m_1 \cdot m_2).$$

(b) Show how the above property of El Gamal leads to a chosen ciphertext attack. That is, assume you are given an El Gamal public key B and a ciphertext (R, S) which is an encryption of some unknown message m and that you are furthermore given

access to an oracle that will decrypt any ciphertext other than (R, S). Based on these things, compute m.

Solution: (Again we implicitly assume computation modulo p.) Since (R, S) is encryption of m there exists an r such that $(R, S) = (g^r, m \cdot B^r)$.

Pick any $m' \neq 1$ and any $r' \neq 0$ and compute

$$R' = R \cdot g^{r'} = g^{r+r'}$$
$$S' = S \cdot m' \cdot B^{r'} = m \cdot m' \cdot B^{r+r'}$$

Submit (R', S') to the oracle for decryption. Note that (R', S') is a valid encryption of $m \cdot m'$ and $(R', S') \neq (R, S)$, so the oracle will give us $m \cdot m'$ as the result. Given $m \cdot m'$, we may simple multiply by m'^{-1} to obtain m.

Question 6 ***Side Channels*** **(7 min)**

A side channel is a channel that leaks information due to the physical implementation. It's a *side* channel in the sense that it is not a theoretical weakness in a system, but rather an effect of its physical implementation. Side channels do not involve two cooperating parties; they instead are used by a single party to extract information they are not meant to have.

(a) Consider implementing the RSA cryptography algorithm. The typical way is to go through the 'key' bit by bit. The pseudo-code looks something like this:

```
foreach (bit in key) {
  if (bit) {
    // do multiplication and all hard work if bit is 1
  }
  // do other simpler stuff that you need to do regardless
}
```

Recall the cable box with a tamper resistant private key inside it that Prof. Paxson talked about in the lecture. Can you imagine a side-channel attack on the above implementation to find the private key? HINT: Can you do something with a multimeter?

Solution: The length of time the power is at its peak can give you a clear indication of the bit pattern of the key. Multiplication usually requires more power, and this is noticeable in embedded systems. For example, see https://en.wikipedia.org/wiki/Power_analysis.