CS 161 Spring 2010 — Computer Security — Paxson/Wagner — Discussion 9

1. Authentication on the web

(a) What are some problems with password authentication?

(b) List pros and cons for each of the following authentication mechanism (relative to passwords).

i. You have to answer *n personal knowledge questions*. Examples of these include: "What is your favorite color?" or "What is the name of your favorite baseball team?".

ii. During password creation, you are presented with a grid of *n* (human) faces *k* times. For each grid, you must select a face. Your password consists of the *k* faces that you choose. Authentication is exactly like creation, but now the face you pick must match the one that you selected during creation. The ordering of the faces within a grid is randomized.

iii. When you need to create your password, you are given a picture, and you must select *n* points in the picture. Those points comprise your password. When it is time to authenticate, the server sends you the picture and you pick the points that comprise your password.

Answer:

(a) Hard to remember passwords, hard to create unguessable passwords, tradeoff between memorability and guessability, keystroke timing attacks, unsafe password storage (people write down passwords sometimes or store them somewhere they shouldn't be stored), password reuse.

(b) Alternate authentication mechanisms

i. *Personal knowledge questions*. **Cons:** Many questions don't have enough entropy (e.g., there are only a handful of favorite colors that people have). Facebook, etc. can reveal answers to a lot of these questions (e.g., your favorite baseball team). Many answers are easily guessable. Notable celebrity pwnage due to personal knowledge questions includes Sarah Palin, Paris Hilton. People don't always answer the same (e.g., maybe your favorite movie changes). **Pros:** Easier to remember than passwords.

ii. *Faces.* **Cons:** Potentially less entropy than traditional passwords. Shoulder surfing (looking over someone's shoulder in order to learn someone's password) is easy. **Pros:** Supposed to be more usable. This paper from USENIX Security 2004 looked at the amount of entropy in this scheme: On User Choice in Graphical Password Schemes by D. Davis, F. Monrose, and M.K. Reiter. (`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.5091&rep=rep1&type=pdf`)

iii. *Points in a picture.* **Cons:** If the picture only has a few "hot spots," the password is easily guessable. If it has a lot of hot spots (or no spot is much hotter than another), then passwords may not be as memorable. **Pros:** User doesn't select the picture, so no password reuse, and pictures can be strategically chosen by the server so that it only serves pictures that have a lot of hot spots. Easier to remember than passwords. A 2005 paper from the Symposium On Usable Privacy and Security looked at the usability of graphical passwords: Authentication Using Graphical

Passwords: Effects of Tolerance and Image Choice by S. Wiedenbeck, J. Waters, J-C Birget, A. Brodskiy, and N. Memon. (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.1346&rep=rep1&type=pdf) This 2007 paper from USENIX Security looks at the issue of hot spots: Human-seeded attacks and exploiting hot-spots in graphical passwords by J. Thorpe and P.C. van Oorschot. (http://portal.acm.org/citation.cfm?id-1362911)

2. Capabilities

(a) In lecture you learned about Access Control Lists for managing users' access permissions to shared files. An alternate approach to managing access to resources is the use of *capabilities*. A capability is an unforgeable pointer to a system resource. In a capability-based system, the capability to a resource both designates the resource and grants authority to access that resource. A typical real-world example of a capability is your car keys. The keys both designate the resource (your car keys only work for your car) and grant access rights (anyone in possession of the keys can start your car).[1] Capabilities are transferrable: just like with your car keys, any entity with a capability for a resource can pass that capability on to any other entity.

Programs written for systems that use ACLs can be vulnerable to a confused deputy attack. A classic example of this is a compiler program that writes the output of the compilation to a file specified by the user and then writes billing information to the BILL file. The user does not have write permission to the BILL file, but the compiler does. If the user specifies BILL as the output file, the compiler will overwrite the contents of BILL with the output of the compilation. This occurs because the compiler has authority to write both files, but the authority is not tied to its purpose or source. This is referred to as *ambient authority*. Explain how capabilities can be used to avoid the problem of ambient authority.

Answer: If the system was purely capability-based then, instead of passing in the name of the output file, the user would pass in a capability to the file it wants to use as the output file. Since capabilities are unforgeable, the user can't pass in a capability to a file unless that user has legitimate access to that file. The compiler program is no longer plagued by the ambient authority problem. It will use its capability for writing to the BILL file and the user's capability for writing the output file.

(b) It is sometimes necessary to revoke a user's permission to access a resource. With an ACL-based system this is done by updating the ACL with the new permissions. However, with capabilities this isn't possible. Imagine Alice and Bob are using a capabilities-based OS. The system maintains a mapping of 128-bit identifiers to system resources. The identifier is the capability for the resource. When a process attempts to access a resource it must present its capability as part of the system call. The OS will lookup the capability in its table and then grant access to the appropriate resource. Alice has a capability that grants her read/write permission to a LOG file. She would like to give Bob this capability, but she knows she will want to revoke it at some later date. What could Alice give Bob that will give him the ability to read and write the LOG file now, but still allow her to revoke that permission at some future date? You can introduce additional resources with trusted processing (that is, each resource will correctly execute what you specify it should do) as long as the result retains the basic properties that capabilities provide.

Answer: Alice can create a proxy process that has a capability for the LOG file. The proxy merely forwards all writes to the LOG file. Alice then passes to Bob a capability for the proxy which he can

[1] Car keys are not a perfect example of capabilities. For one thing, there are ways to start your car even without posession of the keys. A second problem with car keys is that it might be possible to forge a valid set of keys given only the car lock.

use to write to the LOG file. When Alice wants to revoke Bob's capability, she only has to change the proxy so that it no longer forwards writes to the LOG. Bob is free to pass on his capability to anyone he chooses, but since he is only passing on the capability to the proxy, any capabilities he passes on will be revoked at the same time Bob's capability is revoked.

(c) Web-based systems that use ACLs (e.g., username/password/cookies authentication) are also vulnerable to confused deputy attacks (e.g., CSRF attacks). You saw in class how web-based email sites that use cookie-based authentication also use capabilities to prevent CSRF attacks. When a user logs in, the server passes an authentication cookie to the client. Whenever the client requests an HTML page from the server, the server includes a random value in the HTML form of the web page. When the user makes a request, say to delete an email, the request includes the random value from the web form. The server checks that it is the correct random value for the client with the given authentication cookie and, if it is correct, honors the request. Suppose the following URI will delete all the mail in a user's inbox: `https://mymail.webby.com?action=del&folder=inbox/?token=A90..89` where `A90..89` represents the random value sent by the server. If the random value is 64 bits long and `webmail.com` has 250 million users, but only about 10 million users are logged in at any one time, what is the attacker's chance of success for a single CSRF attack? In other words, if one user clicks on the link, what is the probability that the attack succeeds?

Answer: $\frac{1}{2^{64}}$

(d) Now suppose `webmail.com` changes their system to use capability-based authentication instead of username/password authentication. `webmail.com` associates with each user's account a capability that grants access to that account. Now the user does not need to provide a username and password, they just need to provide the correct URI. For example, the URI `https://mymail.webmail.com?action=view&folder=inbox/?token=A90..89` will grant the requester permission to view the inbox. The random value `A90..89` is a static value associated with that user's account. If the random value is 64 bits long, what is the attacker's chance of success for a single CSRF attack?

Answer: Roughly $\frac{2^{28}}{2^{64}}$. The attacker only has to guess one of the 250 million valid capabilities currently assigned to users.

3. **Detection strategies** Suppose you are building an Network Intrusion Detection System (NIDS) for the corporate network you run. In particular, you are concerned about malicious modification or deletion of files in the directory `/var/secret/MacGuffin/`.

(a) One method of detection is called "signature matching." This involves looking for particular well-defined patterns in traffic that are known to represent malicious activity. Give a couple of examples of signatures you can use to detect these attacks. What are some limitations of this approach?

Answer: Example signatures:

i. Look for the string "`/var/secret/MacGuffin`" in requests
ii. Look for "`rm -rf`"
iii. Wait until a particular attack occurs. Afterwards, look for the same packets as occurred during that attack.

Problems with this approach:

i. It is prone to false positives as it lacks context. It could be that access to `/var/secret/MacGuffin` occurs frequently for benign reasons, and without ensuing modifications. Similarly, users might often use `rm -rf` to manipulate directories other than the one you're observing.

ii. It can be prone to false negatives or evasion. For example, an interact attacker could issue `cd /var/secret; cd MacGuffin` followed by `rm -f -r .` and easily evade detection.

iii. Note that if you literally only add matches based on known (= previously seen) attacks, then the approach is purely reactive; if you're looking for a threat unique to your site, you cannot inoculate yourself from it until you have suffered it. On the other hand, (1) if the threat is one faced by other sites, they might have written signatures for it after having experienced it, and (2) one can adapt signature technology to write *vulnerability* signatures (signatures that match a known potential problem, rather than a known specific attack), which *can* be proactive.

(b) Another approach is to search for behaviors. Instead of looking for known attacks, the detector might use knowledge of the system to look for suspicious sets of actions. Give two examples of host-based behavioral detection. Be specific as to how your example differs from signature matching that looks for known attacks. What are some problems with this approach?

Answer: Examples:

i. Look for the removal of files in `/var/secret/MacGuffin` after multiple attempts at logging in as "root". Here, rather than looking for a specific attack we're looking for a pattern associated with likely-attack activity.

ii. Don't even look for attacks; look for related suspicious activity indicative of a compromise. For example, look for attempts at accessing and deleting log files over an SSH connection. This approach can potentially detect a wide range of compromises during which the attacker obtains login access to the target system.

Issues:

i. Requires lots of parsing in order to understand many protocols. This is potentially a lot of work.

ii. While potentially more general than signature matching, can still miss a wide range of attacks that don't happen to include (or for which the attacker consciously avoids including) the behavior for which we monitor.

(c) Suppose now we aim to detect modifications to any files in `/var/secret/MacGuffin` using the following procedure. Each night, we run a cron job that checksums all of the files in the directory using a cryptographically strong hash like SHA256. We then compare the hashes against the previously stored ones and alert on any differences. (This scheme is known as "Tripwire.")

Discuss issues with false positives and false negatives.

Answer: False positives can occur any time that the files are changed for a legitimate purpose.

False negatives should not be a direct problem: due to the properties of a hash function like SHA256, if an attacker makes any modification to a file, the hash will change; they will not be able to find any alternative value for the file that yields the same hash.

However, if the attacker gains administrative privileges then they could modify the OS to return the old content of the file whenever the nightly job runs; or modify the nightly job directly to always report nothing has changed; or modify the stored hashes to reflect the new content of the file.

(d) Continuing the previous scenario, suppose the attacker was able to subvert the operating system. Can you think of a procedure (which might be expensive in terms of labor) by which an operator could still detect the modified files?

Here's one approach that has been used in practice. The hashes aren't stored locally but instead on a remote system (which prevents the attacker from tampering with them). When the operator wants to check a file system, they shut down the suspect machine and remove the disk, mounting it on a separate system (with a presumably trustworthy OS) for comparison. Alternatively, the operator could insert a boot disk into the suspect machine and boot off of read-only media (assuming the attacker cannot alter the low-level boot sequence) and use that alternative OS for the validation procedure.