Paxson
Spring 2013

CS 161 Computer Security

Discussion 12

April 17, 2013

Question 1 ***Detection strategies*** **(10 min)**

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/log/sonny_dykes/`.

(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?

Solution: Example signatures:

1. Look for the string "`/var/log/sonny_dykes`" in requests
2. Look for "`rm -rf`"
3. Wait until a particular attack occurs. Afterwards, look for the same packets as occurred during that attack.

Problems with this approach:

1. It is prone to false positives as it lacks context. It could be that access to `/var/log/sonny_dykes` 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.
2. It can be prone to false negatives or evasion. For example, an interact attacker could issue `cd /var/log; cd sonny_dykes` followed by `rm -f -r .` and easily evade detection.
3. 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?

Solution: Examples:

1. Look for the removal of files in `/var/log/sonny_dykes` 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.
2. 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:

1. Requires lots of parsing in order to understand many protocols. This is potentially a lot of work.
2. 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/log/sonny_dykes` 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.

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

Given a single change to a file, 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.

In addition, if the attacker makes a change to the file to their benefit, but then *changes the file back* prior to the run of the nightly job, then they will escape detection (false negative).

(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?

Solution: 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.

Another approach used in practice is for the security analyst to copy a self-contained environment providing key system diagnosis tools over to the compromised system. This is unsound if the modified OS detects its presence and subverts its operation, but often can provide benefit in practice because the subverted kernel in fact does not particularly look for it; the approach hinges on the assumption that rootkits exclusively mess with the installed toolchain and do not bother with such custom environments. The `busybox` environment, for example, provides numerous tools that replace classic targets for subversion, such as `ls`, `ps`, `find`, `grep`, `mount`, `ifconfig`, and many more.

Question 2 *Detecting Web Attacks* (10 min)

At this year's annual *Grasses For The Masses* home & garden convention, in beautiful Fairfax California, the startup *Lazer Lawns*—which specializes in producing so-juicy-looking-you-just-wanna-eat-it artificial turf—experienced a live SQL injection attack from the audience while showcasing their new high-end collection of silver-ionized heat-repellent blades—what a disaster! After firing the organizer of the event and hiring an CS161-educated security expert, Grasses For The Masses now plans to install a NIDS that watches the free WiFi next year. Moreover, Lazer Lawns has learned the hard way to make sure they have a HIDS protecting their assets.

As a potential advisor to either Grasses For The Masses or Lazer Lawns, consider the most prevalent web attacks: XSS (both reflected and stored), CSRF, and SQL injection injection.

(a) For each attack, devise one or more concrete strategies based on signature, behavioral, or anomaly detection. Include a discussion of false positives and false negatives.

Solution: XSS. In order to detect *reflected* XSS, one can look for `<script>` tags in the URI of HTTP GET requests. Since there exist a multitude of different encodings and syntactic variations (as well as other contexts in which a reply will be interpreted as Javascript), this *signature-based* scheme is particularly vulnerable to evasion and may exhibit a high number of false negatives.

In principle, one could detect *stored* XSS by inspecting the body of HTTP POST requests for script content (that is, detect the injected script when it is uploaded, rather than when the server subsequently sends it to the victim). Again, one might scan for `<script>` tags, with the same considerations as above likewise applying.

Also in principle one can detect reflected XSS by looking for substrings in HTTP requests that reappear in HTTP replies. This approach however may have significant false positives, depending on the structure of the web service.

CSRF. For HTTP requests with a cookie, a very basic *behavioral* CSRF attack detector could check that the domain name in the `Referer` matches the domain name of the server, e.g., by looking for a past server response with a corresponding `Host` header. This approach is prone to false positives since browsers and extensions have full control over the content of the `Referer` header (in particular, they can omit it or reset it to the empty string), and thus render detection impossible.

SQL injection. One might consider employing *specification-based* detection, where the specification requires only a constrained set of characters (which do not include any SQL meta-characters) from appearing in requests that the server receives. The effectiveness of this approach will depend on whether the specification is viable, i.e., doesn't rule out any legitimate traffic that the server needs to support. It also depends on the ability to correctly identify the constrained set and whether it indeed suffices to prevent an attacker from constructing a successful SQL injection.

A more robust approach would involve inferring the SQL that will result from a given user input, parsing it as SQL, and making a pass over the AST to perform semantic analysis.

(b) Explain whether a network-based or host-based deployment approach makes more sense for your devised detection strategy (or if it doesn't really matter). Does the deployment angle have an effect on your detection rates?

Solution: In all three attack scenarios, if Lazer Lawns employs HTTPS it makes more sense to deploy as a HIDS because the NIDS at Grasses For The Masses will not have the private key to decrypt and inspect the HTTP traffic.

In addition, a HIDS will not face as many issues regarding evasion threats as a NIDS, because it analyzes information right at the potential victim. In some cases, a HIDS can leverage a richer understanding of the application data. For example, a database could parse a SQL query and hand the AST over to a detection module that determines whether continuing is safe. This approach also has the advantage of avoiding ambiguities at the network level, such as those retransmission-based scheme we looked at in lecture.

Question 3 ***Technical expert for the defense*** **(optional)** **(10 min)**

Bob is being prosecuted for sending spam email, in violation of US law. The prosecutor has accumulated a variety of evidence against him: there is irrefutable evidence that many spam emails were sent from an IP address that Bob's ISP says was allocated to him at the time; when the cops seized Bob's laptop, they found evidence in Bob's browser history that his machine had been used to visit web sites explaining how to make money off of spam; when Bob was interviewed, he admitted that the laptop was his personal laptop and that no one else had access to it, to his knowledge. The prosecutor states that Bob had the knowledge, the skill, and the means to commit the crime, and accuses him of it.

Bob vehemently denies the accusations. He admits that he visited the web site, but says he was just curious about how the spam underground worked, and says that he never sent the spam email in question. Bob's lawyer has hired you as a technical expert for the defense. You've interviewed Bob, and you have decided to start from the assumption that Bob is telling the truth as he sees it. You have no reason to disbelieve the prosecutor's representation of the evidence.

List two possible scenarios that are compatible both with Bob's claims and with the evidence listed by the prosecutor. For each scenario, explain why it exonerates Bob.

Solution:

- Bob's laptop might have been infected by malware, which took control of his machine and used it to send spam.
- Similar: the hacker web sites might have added malicious stuff to their web page to exploit vulnerabilities in Bob's browser, so that when Bob visited the hacker web sites, unbeknownst to him, his laptop was infected.
- Bob has a wireless router at home, and it's an open wireless network. A spammer drove by, noticed the open network, hopped on it, and sent spam.
- An insider at the ISP sent the spam himself, and altered the records of who the source IP address was assigned to at the time.