Paxson
Spring 2017

CS 161
Computer Security

Discussion 12

Week of April 24, 2017

Question 1 ***Detection strategies*** **(20 min)**

Suppose you are responsible for detecting attacks on the UC Berkeley network, and can employ host-based monitoring (a HIDS) that can inspect the keystrokes that users enter during their shell sessions. One particular attack you are concerned with is malicious modification or deletion of files in the directory `/usr/oski/config/`.

(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 "`/usr/oski/config/`" 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 `/usr/oski/config` 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 attacker could issue `cd /usr/oski; cd config` followed by `rm -f -r .` and easily evade detection.
3. If you only create signatures 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 examples differ from signature matching that looks for known attacks. What are some problems with this approach?

Solution: Examples:

1. Look for changes to files in `/usr/oski/config/` 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 login sessions that immediately issue `rm` commands, based on knowledge that benign users don't start off their sessions by removing files, but those who want to mess with Oski very well might. This approach can potentially detect a wide range of compromises for which the attacker obtains login access to the target system.

Issues:

1. Relies on the assumption that benign users will very rarely exhibit the behavior we key off of.
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 `/usr/oski/config/` 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* (15 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 a 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[1] protecting their assets.

As a potential advisor to either Grasses For The Masses or Lazer Lawns, consider the some

[1] Given they have to demo their software in many different environments, they've learned that they shouldn't rely on being able to employ a NIDS, hence their emphasis on using a HIDS.

prevalent web attacks: XSS (both reflected and stored) and SQL injection.

(a) For each attack, devise one or more concrete strategies based on signature, behavioral, anomaly, or specification-based 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 URL 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.

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.

(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 each attack scenario, 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 an SQL query and provide the resulting Abstract Syntax Tree (AST) 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 the retransmission-based evasion we looked at in lecture.

Question 3 ***Detection Tradeoffs*** **(15 min)**

Suppose that S is a network-based intrusion detector that works by passively analyzing individual UDP and TCP packets. Suppose that A is a host-based intrusion detector that is a component of the browser that processes and analyzes individual URLs before they are loaded by the browser. Suppose S has false positive rate S_P and false negative rate S_N, and A has false positive rate A_P and false negative rate A_N.

Your company decides to build a hybrid scheme for detecting malicious URLs. The hybrid scheme works by combining scheme S and scheme A, running both in parallel on the same traffic. The combination could be done in one of two ways. Scheme H_E would generate an alert if for a given network connection either scheme S or scheme A generates an alert. Scheme H_B would generate an alert if both scheme S and scheme A generate an alert for the same connection. (Assume that there is only one URL in each network connection.)

(a) Assuming that decisions made by S and A are well-modeled as independent processes, and ignoring any concerns regarding evasion, what can you say about the false positives and false negatives of H_B and H_E? In terms of S_P, S_N, A_P, A_N, what are the false positive and false negative rates for H_B and H_E?

(b) If deploying the hybrid scheme in a new environment, is one of H_E and H_B clearly better? If so, which one, and why? If not, what environment parameters would help determine whether H_E or H_B is better?

Solution:

(a) The key insight here is that alarms by H_B will be a subset of the alarms generated by H_E. Since H_B will generate fewer alarms for non-malicious activities, it will have less false positives. On the other hand, because it generates fewer alarms, it will likely miss more malicious activity, leading to more false negatives. The false positive rate for H_E would be $S_P + A_P - S_P A_P$, and for H_B would be $S_P A_P$. Similarly, the false negative rate for H_E would be $S_N A_N$ and for H_B would be $S_N + A_N - S_N A_N$

(b) In the absence of more data, particularly the cost of false positives and false negatives, as well as the rate of malicious and non-malicious activity, it is impossible to make a definitive decision.

Increasing the cost of false positives, and the rate of non-malicious activity, favors scheme H_B. Increasing the cost of false negatives, and the rate of malicious activity, favors scheme H_E.