Intrusion Detection Systems

Sources:

- *Computer System Intrusion Detection: A Survey* by A. K. Jones and R. S. Sielken
- *Research in Intrusion-Detection Systems: A Survey* by S. Axelson
- *Security Warrior* by C. Peikari and A. Chuvakin
- *Modern Intrusion Detection, Data Mining, and Degrees of Attack Guilt* by S. Noel, D. Wijesekera, and C. Youman
- *NetSTAT: A Network-based Intrusion Detection Approach* by G. Vigna and R. A. Kemmerer
- *Network Intrusion Detection Signatures* by K. K. Frederick, www.securityfocus.com
- *Bayesian Event Classification for Intrusion Detection* by C. Kruegel *et al*
- *A Stateful Intrusion Detection System for World-Wide-Web Servers* by G. Vigna, *et al*
- *A Survey of Coordinated Attacks and Collaborative Intrusion Detection* by C.V. Zhou et al

Intrusion detection systems (IDSs) are intended to discover intrusions or attacks against a computer system.

Those that work *online* can counteract an intrusion that is underway and possibly help to identify the perpetrator.

Those that work *offline* can be used to initiate recovery procedures and/or to identify vulnerabilities for repair.

Intrusion detection systems analyze *network packets*, *event logs*, or other types of *execution profiles*.

They generate *alarms* and may also initiate *countermeasures*.

Alternative scopes for IDSs:

- Network based (NIDS)
- Host-based
- Application-based

Host-based IDSs cannot easily detect intrusions involving patterns of distributed activity.

Network-based IDSes typically scan network packets at multiple machines.

Host-based systems look at system call traces.

Application-based IDSs consider application semantics.

Two basic types of detection mechanisms:

- Misuse detection/signature detection
- Anomaly detection

There are also hybrid approaches.

Misuse Detection

Misuse detection systems look for behavior that is characteristic of a *known attack type*.

Example: oversized string argument sent to particular port

Example: The *phf* exploit involves an HTTP request like the following:

```
http://host.com/cgi-bin/phf?
    Qalias=%0A/bin/cat%20/etc/passwd
```

Misuse detection systems can detect known attacks with few false alarms (false positives).

They are *poorly suited* to detecting *novel* attacks.

Major approaches to misuse detection:

- Signature analysis
- Expert systems
- State-transition analysis
- Data mining
- Machine learning

Signature Analysis

An IDS that does signature analysis maintains a database of known *attack signatures*.

Example intrusion signatures [Frederick]:

- Connection attempt from a reserved IP address.
- Packet with an illegal TCP flag combination.
- Fixed email containing a particular virus.
- DNS buffer overflow attempt contained in the payload of a query.
- Denial of service attack on a POP3 server caused by issuing the same command thousands of times.
- File access attack on an FTP server by issuing file and directory commands to it without first logging in.

Intrusion signatures may be described in a formal language supported by an IDS.

Example: *Snort* signature for *phf* exploit [Peikari and Chuvakin]

```
alert tcp $EXTERNAL any -> $HTTP_SERVERS $HTTP_PORTS
(msg: “WEB-CGI phf access”; flow:to_server, established;
uricontent: “\phf”; nocase; reference: bugtraq, 629;
reference: arachnids, 128; reference: cve, CVE-1999-0067;
classtype: web-application-activity; sid: 886; rev: 8;)
```

Some research has been done on generalizing or *abstracting* intrusion signatures to catch a wider class of attacks.

Expert Systems

Rule-based IDSs encode intrusion scenarios as a set of rules.

The system state is represented in a *knowledge base* consisting of a *fact base* and a *rule base*.

A *fact base* is a collection of assertions that can be made based on accumulated data from audit records or from system activity monitoring.

The *rule base* contains rules that describe known intrusion scenarios.

When the pattern of a rule's antecedent (precondition) matches asserted facts the rule *fires*, which initiates an alarm or countermeasure.

Example: Illegal privileged account access rule [Jones and Sielken]:

```
(defrule illegal_privileged_account states
    if there exists a failed_login_item
        such that name is ("root" or "superuser" or "maintenance"
            or "system) and
        time is ?time_stamp and
        channel is ?channel
    then
        (print "WARNING: ATTEMPTED LOGIN TO PRIVILEGED
            ACCOUNT")
        and remember a breakin_attempt
        with certainty *high*
        such that attack_time is ?time_stamp
        and login_channel is ?channel)
```

State-Transition Analysis

In state-based representations, *attribute-value pairs* characterize system states of interest.

The *state* of a system is a function of all the users, processes, and data present at a given time.

Actions that contribute to intrusion scenarios are characterized as *transitions* between states.

Each action changes the value of attribute(s) of interest.

Intrusion scenarios are characterized by *state transition diagrams*.

Nodes represent system states and arcs represent actions.

Some states are designated *compromised*.

Example: Finite state diagram for cookie stealing [Vigna, *et al*]:

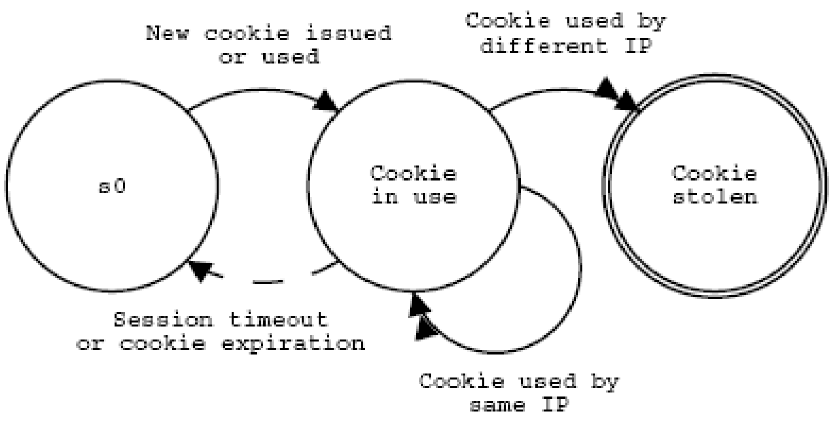
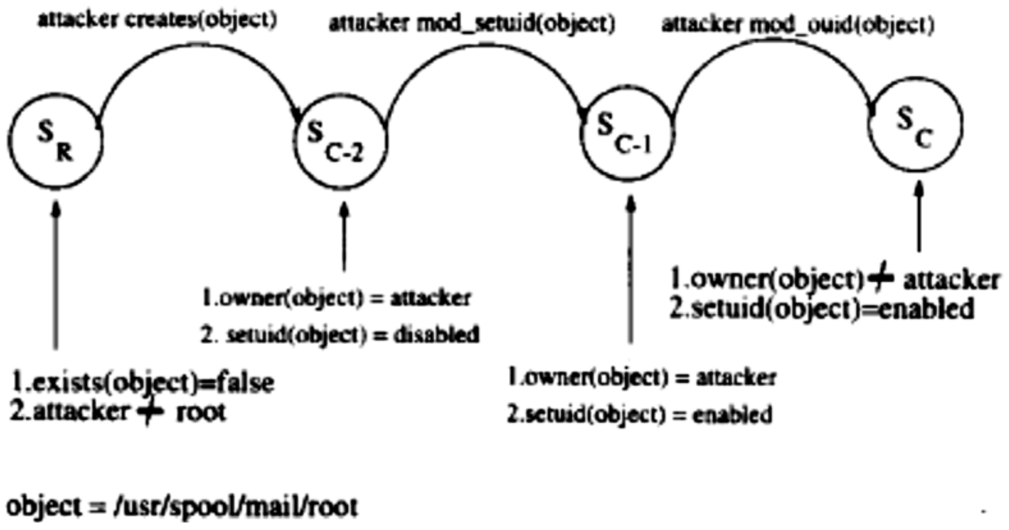

New cookie issued or used
Cookie used by different IP
s0
Cookie in use
Cookie stolen
Session timeout or cookie expiration
Cookie used by same IP

Figure 4. State-transition diagram for the cookie stealing scenario.

Example: FSM for BSD UNIX penetration scenario [Ilgun, et al, IEEE TSE 21, 3, 1995]:


attacker creates(object)
attacker mod_setuid(object)
attacker mod_ouid(object)
S_R
S_{C-2}
S_{C-1}
S_C
1.exists(object)=false
2.attacker ≠ root
1.owner(object) = attacker
2. setuid(object) = disabled
1.owner(object) = attacker
2.setuid(object) = enabled
1.owner(object) ≠ attacker
2.setuid(object)=enabled
object = /usr/spool/mail/root

Data Mining Approach to Misuse Detection

This involves training a *pattern classifier* to recognize known attacks.

The classifier may be statistical, a learning algorithm, or based on association rules.

It requires a *training set* consisting of attack instances mixed with normal executions.

Anomaly Detection

Anomaly detection systems distinguish between normal and abnormal (anomalous) system state or behavior.

Static Anomaly Detection

A *static* anomaly detector is based on the assumption that there is a portion of the system state being monitored that should remain constant.

If this portion changes, the detector raises an alarm.

Example: *Tripwire* detects changes in file attributes that are expected to remain constant.

Dynamic Anomaly Detection

A *dynamic* anomaly detector raises an alarm when it observes a significant deviation from normal system behavior.

Dynamic anomaly detectors typically create a *base profile* to characterize normal, acceptable behavior.

A profile consists of a set of *observed measures* of behavior for each of a set of dimensions, e.g.:

- Preferred choices, e.g., log-in time, log-in location, favorite editor

- Resources consumed cumulatively or per unit time, e.g., length of interactive session or number of messages sent per unit time

- Representative sequences of actions

Dimensions may be specific to the type of entity with which behavior is associated.

Typical entities are users, workstations, remote hosts, or applications.

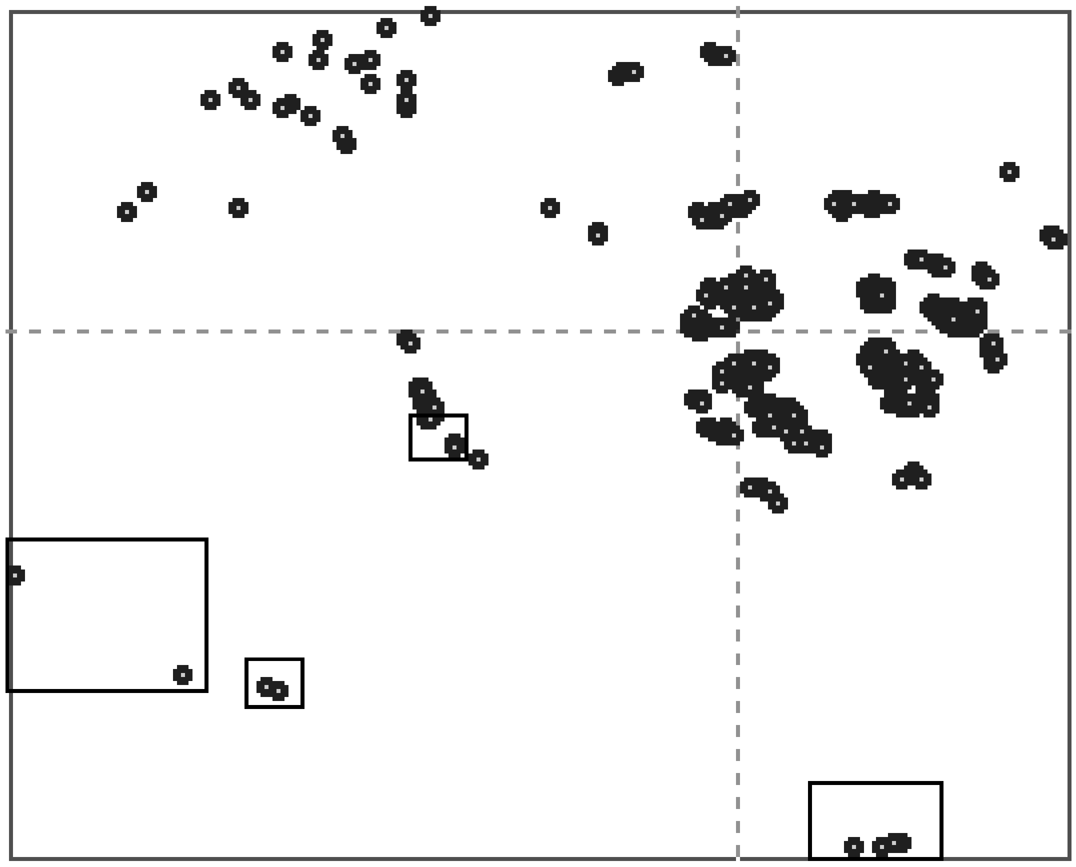
Example: Forrest *et al* created an anomaly detector that examines short sequences of (6-10) OS calls.

These are called *N*-grams.

The detector raises an alarm when a sequence that has not been seen before occurs.

Such an IDS can be evaded by “padding” the system call sequence with irrelevant calls.

Example: *Multidimensional scaling* plot of JBoss information flow profiles [Masri and Podgurski]:



Attacks are shown in boxes.

Example: Bayesian network classifier [Kruegel]:

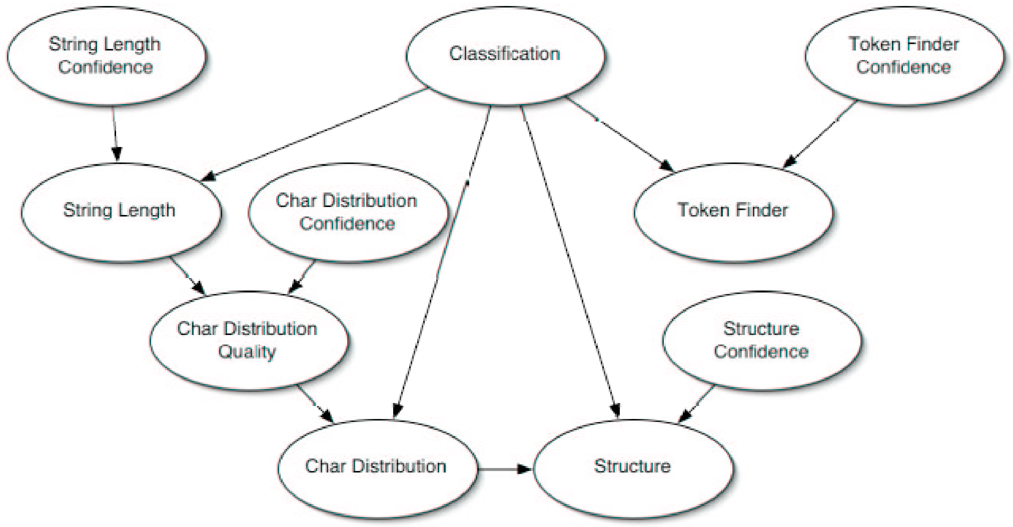


Figure 3. Bayesian Network for `open` and `execve` System Calls

Anomaly detection systems are more vulnerable to *false alarms* (false positives) than misuse detection systems.

This is because unusual behavior does not necessarily indicate an attack.

"Base rate fallacy" occurs when the conditional probability of the hypothesis "attack" given evidence *E* is assessed without taking into account

- The prior probability (base rate) of attack; and
- The total probability of evidence *E*

$$\Pr[attack|alarm] = \frac{\Pr[alarm|attack]\Pr[attack]}{\Pr[alarm]}$$
$$= \frac{0.99 \times 0.001}{0.01} = 0.099$$

Coordinated Attacks and Cooperative Intrusion Detection

Recent attacks target or utilize a large number of hosts spread over a wide geographical area or multiple administrative domains.

These are called *large-scale coordinated attacks*.

They may be difficult to detect since the evidence is spread over multiple domains.

Example [Zhou et al]: *stealthy scan*

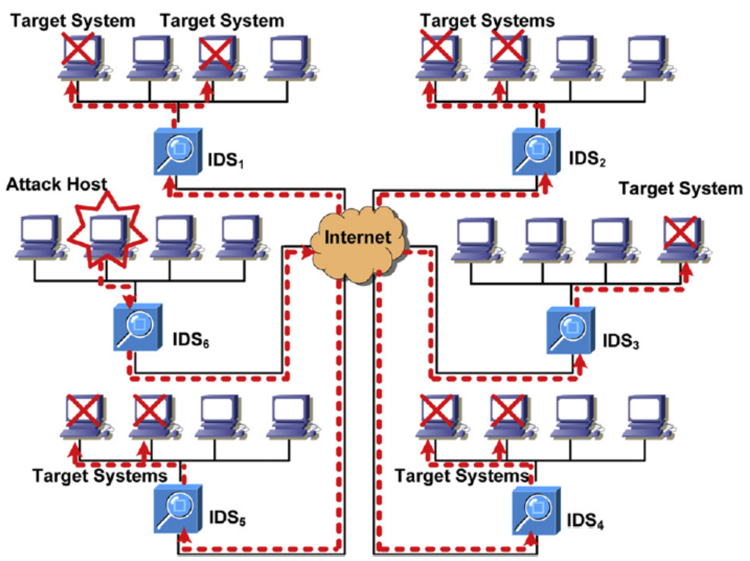


Fig. 1 – A common stage of coordinated attacks.

Typical attack process:

1. *Large-scale stealthy scans* used to hunt for known vulnerabilities or list of vulnerable addresses

2. Attacker writes a *worm script* to compromise many hosts simultaneously

3. Attacker launches *large scale DDoS attacks*

Stealthy scans *randomize the scan order* or use a *low scan frequency* to evade detection.

A worm is a self-replicating program that can send copies of itself to other computers in the network.

Worm spread has two phases: *infection* and *propagation*.

Distributed denial-of-service (DDoS) attacks use *multiple attack sources* to amplify the power of the attack.

A *distributed reflector attack* uses innocent third-party systems (routers or servers) to relay attack traffic to the target and obscure its sources.

I*Reflectors*: IP hosts that will return a packet if sent one.

The attacker instructs its zombies to send the attack traffic to the reflectors with the target's IP address as the source address.

Collaborative IDSs

A CIDS has two main functional units:

- *Detection unit* – consists of multiple detection sensors, each monitoring its own subnetwork
- *Correlation unit* – transforms low-level alerts into high-level intrusion report of confirmed attacks

CIDS system architecture may be centralized, hierarchical, or distributed.

In the *centralized approach*, each IDS plays the role of a detection unit in the CIDS.

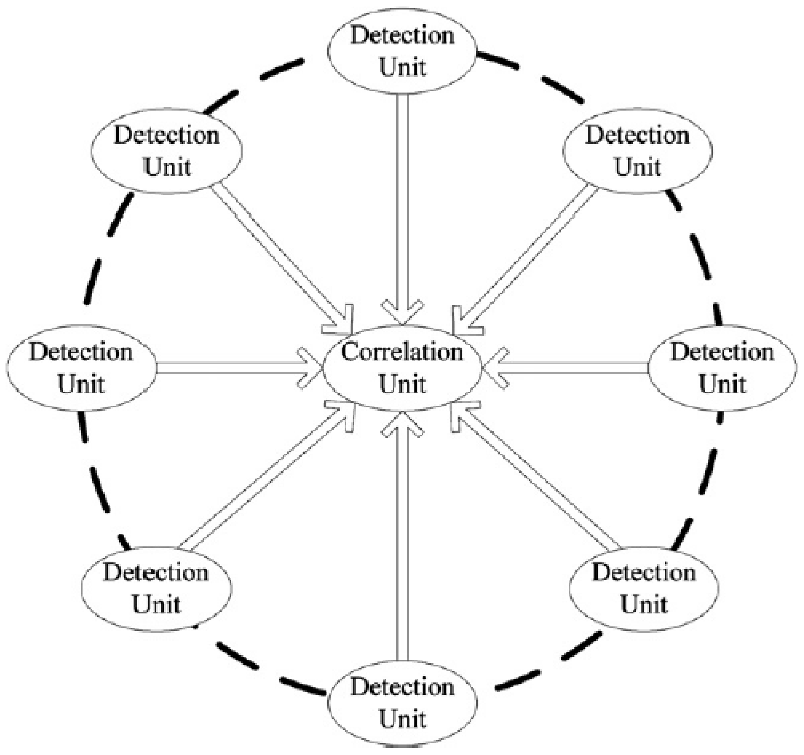


Fig. 2 – Centralized CIDS architecture.

Alerts are reported to a central server that works as a correlation unit for analysis.

Limitations of centralized approach:

- Central unit is a *single point of failure* or vulnerability
- Processing capability of central node *limits the volume of data* it can handle.

In the *hierarchical approach*, the CIDS is partitioned into small *communication groups* based on

- Geography
- Administrative control
- Collection of similar software platforms
- Anticipated types of intrusions

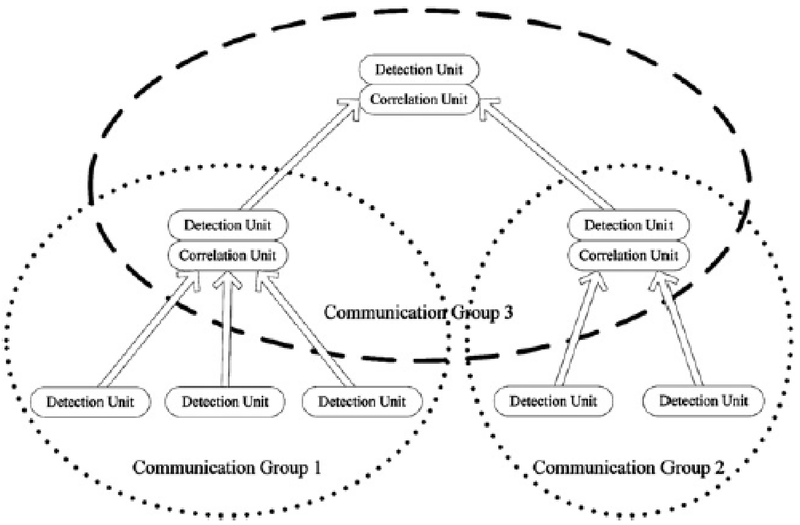


Fig. 3 – Hierarchical CIDS architecture.

There is an *analysis node* in each group that is responsible for correlating data collected in the group.

Its process data is sent upward to a node further up in the hierarchy for analysis.

IDSs in the lowest level act as detection units.

IDSs in the higher level(s) have *both* detection and correlation units.

In a *fully distributed CIDS*, each participating IDS has both a detection unit and a correlation unit.

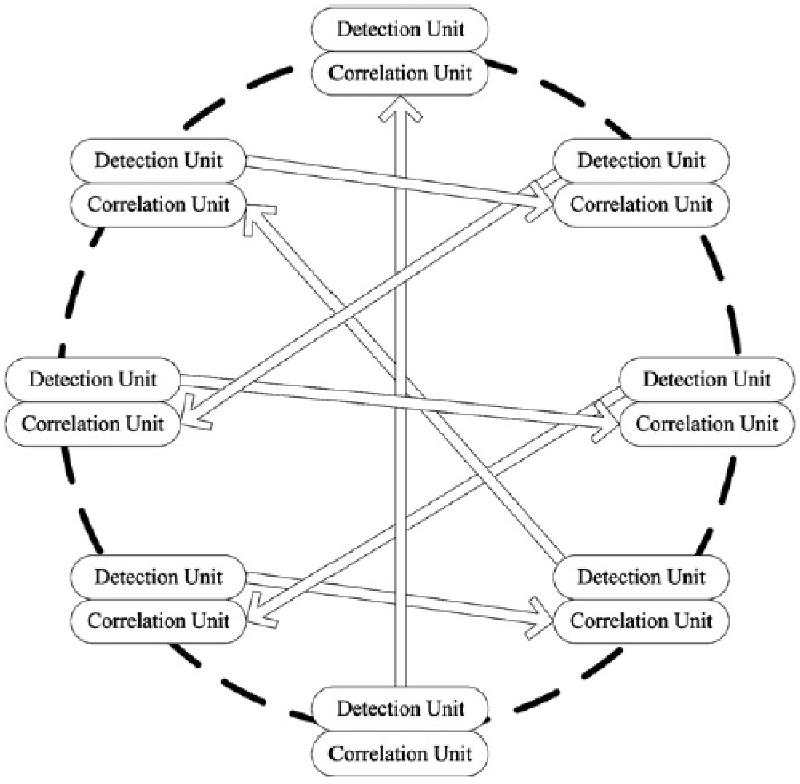


Fig. 4 – Fully distributed CIDS architecture.

The detection unit is responsible for collecting data locally.

The correlation unit is part of a *distributed correlation scheme*.

The participant IDSs communicate using some form of *data distribution protocol*, e.g.,

- Peer-to-peer (P2P)
- Gossiping
- Multicast
- Publish/subscribe

Alert Correlation

CIDS alert correlation techniques can be classified into four groups:

- *Similarity based* – calculate similarity between alerts based on their attributes
- *Attack-scenario based* – attempt to recognize predefined attack scenarios
- *Multi-stage* – assume attackers perform multiple steps to fulfil global intrusion plan
- *Filter based* – prioritize alerts by their impact to protected systems