NC STATE UNIVERSITY Computer Science

CSC 774 Network Security

Topic 5.1 Intrusion Alert Correlation

Outline

- Why do we need to correlate intrusion alerts
- Approaches to intrusion alert correlation
- Alert correlation based on prerequisites and consequences of attacks
  - A Formal Framework for Alert Correlation
  - Implementation
  - Experimental Evaluation

Why Do We Need Alert Correlation?

- CERT's overview of attack trends (04-18-02)
  - Increasing automation
  - Increasing sophistication of attack tools
- Traditional intrusion detection systems (IDS)
  - Focus on low-level attacks or anomalies
  - Mix actual alerts with false alerts
  - Generate an unmanageable number of alerts
    - ID practitioners: "Encountering 10,000 to 20,000 alerts per day per sensor is common"
- We need automated tools to…
  - construct attack scenarios
  - facilitate intrusion analysis

Approaches to Alert Correlation

- Method 1: Exploit similarities between alert attributes
  - Ex.: Valdes and Skinner (2001), Staniford et al. (2000)
  - Limitation: Cannot fully discover the causal relationships between alerts
- Method 2: Exploit known attack scenarios
  - Ex.: Cuppens and Ortalo (2000), Dain and Cunningham (2001), Debar and Wespi (2001)
  - Limitation: Restricted to known attack scenarios or those generalized from known scenarios

Approaches to Alert Correlation (Cont'd)

- Method 3: Use prerequisites and consequences of attacks
  - JIGSAW by Templeton and Levitt (2000)
    - Cannot deal with missing detections and failed attacks
  - MIRADOR approach by Cuppens and Miege (2002)
  - TIAA approach by Ning, Cui, and Reeves (2002)
    - Tolerate missing detections and false alerts (Compared with JIGSAW)
    - Allow flexible manipulation after correlation (Compared with MIRADOR approach)
    - Developed independently and in parallel to MIRADOR approach.

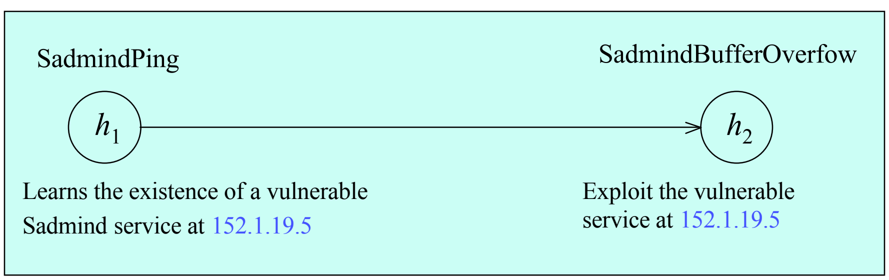
The TIAA Approach

- Related Papers
  - Peng Ning, Yun Cui, Douglas S. Reeves, "Constructing Attack Scenarios through Correlation of Intrusion Alerts," in Proceedings of the 9th ACM Conference on Computer & Communications Security, pages 245--254, Washington D.C., November 2002.
  - Peng Ning, Yun Cui, Douglas S. Reeves, "Analyzing Intensive Intrusion Alerts Via Correlation," in Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002), LNCS 2516 , pages 74--94, Zurich, Switzerland, October 2002.
  - Peng Ning, Dingbang Xu, "Adapting Query Optimization Techniques for Efficient Intrusion Alert Correlation," Technical Report TR-2002-13, Department of Computer Science, North Carolina State University, August 2002.

A Formal Framework for Alert Correlation

- Represent our knowledge about individual types of attacks
  - Prerequisite: *necessary* condition or system state for an intrusion to be successful
  - Consequence: *possible* outcome or system state of an intrusion
    - Must be true if the intrusion succeeds.
- Correlate alerts (i.e., detected attacks) by reasoning about the consequences of earlier attacks and the prerequisites of later ones

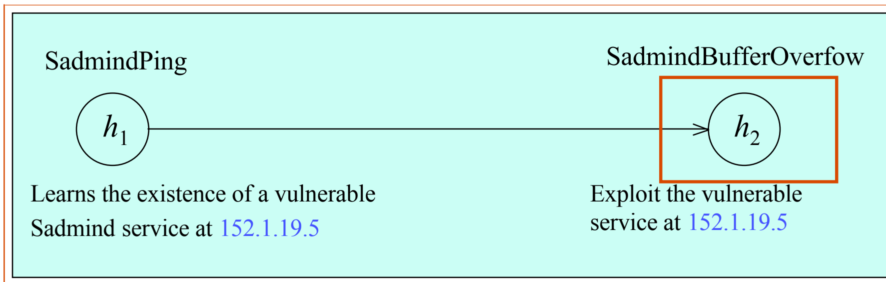
Example



A Formal Framework (Cont'd)

- Use predicates to represent system state or attacker's knowledge.
- A hyper-alert type *T* is a triple (*fact*, *prerequisite*, *consequence*)
  - *fact* is a set of attribute names
  - *prerequisite* is a logical combination of predicates whose free variables are in fact
  - *consequence* is a set of predicates s.t. all free variables in consequence are in fact

Example



- Example
  - SadmindBufferOverflow=({VictimIP, VictimPort}, ExistHost(VictimIP)^VulnerableSadmind(VictimIP), {GainAccess(VictimIP)})
  - This is the knowledge about a type of attacks, not attack instances.

A Framework (Cont'd)

- How to represent IDS alerts (detected attacks)?
- Given a hyper-alert type *T* = (*fact*, *prerequisite*, *consequence*), a hyper-alert (instance) *h* of type *T* is a finite set of tuples on *fact*, where each tuple is associated with an interval-based timestamp [*begin_time*, *end_time*].
  - Allow aggregation of the same type of hyper-alerts.
- Question: Why "a finite set of …"?

Example

- A hyper-alert *h* of type *SadmindBufferOverflow*:
  - {(VictimIP=152.1.19.5, VictimPort=1235), (VictimIP=152.1.19.7, VictimPort=1235)}
  - Prerequisite: ExistHost(152.1.19.5)^VulnerableSadmind(152.1.19.5) and ExistHost(152.1.19.7)^VulnerableSadmind(152.1.19.7) must be True for them to succeed.
  - Consequence: GainAccess (152.1.19.5) and GainAccess (152.1.19.7) might be True, depending on the success of the corresponding attacks.

Correlation of Alerts

- How about requiring the prerequisite of an alert be fully satisfied to correlate it with an earlier set of alerts.
  - Attacker may not always launch earlier attacks to fully prepare for later ones
  - Missing detections
  - Computationally expensive to check
- Our solution
  - Partial match: Correlate two alerts if the earlier attack may *contribute* to the later one.

A Formal Framework (Cont'd)

- Given a hyper-alert type *T* = (*fact*, *prerequisite*, *consequence*),
  - The prerequisite set (or consequence set) of *T* is the set of all predicates that appear in *prerequisite* (or *consequence*)
  - Denoted as P(T) (or C(T))
- Example
  - SadmindBufferOverflow=({VictimIP, VictimPort}, ExistHost(VictimIP)^VulnerableSadmind(VictimIP), {GainAccess(VictimIP)})
  - P(T) =____________________
  - C(T)=____________________

A Formal Framework (Cont'd)

- Given a hyper-alert instance *h* of *T*,
  - The prerequisite set (or consequence set) of *h* is the set of predicates in P(T) (or C(T)) whose arguments are replaced with the corresponding attribute values of each tuple in *h*.
  - Denoted P(h) (or C(h)).
  - Each predicate in P(h) or C(h) inherits the timestamp of the corresponding tuple.

Example

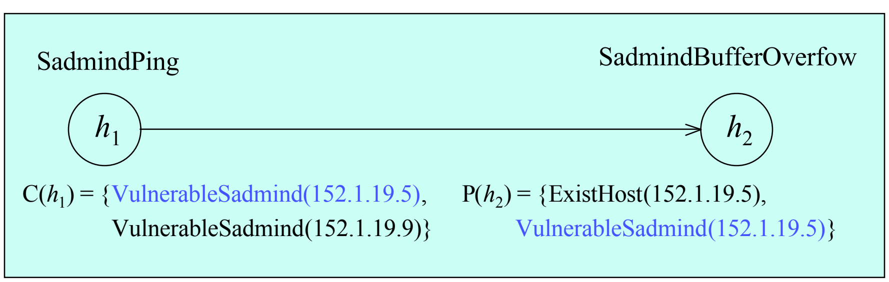
- A hyper-alert *h* of type *SadmindBufferOverflow*:
  - {(VictimIP=152.1.19.5, VictimPort=1235), (VictimIP=152.1.19.7, VictimPort=1235)}
  - Prerequisite: ExistHost(152.1.19.5)^VulnerableSadmind(152.1.19.5) and ExistHost(152.1.19.7)^VulnerableSadmind(152.1.19.7) must be True for them to succeed.
  - Consequence: GainAccess (152.1.19.5) and GainAccess (152.1.19.7) might be True, depending on the success of the corresponding attacks.
  - P(h)=______________________
  - C(h)=______________________

A Formal Framework (cont'd)

- Hyper-alert h_1 *prepares for* hyper-alert h_2 if there exist $p \in P(h_2)$ and $C \subseteq C(h_1)$ s.t.
  - For all $c \in C$, c.end_time < p.begin_time, and
  - The conjunction of the predicates in C implies p.
- Intuition: h_1 prepares for h_2 if some attacks represented by h_1 make some attacks represented by h_2 easier to succeed.

Example

- Intuition of correlation
  - An earlier hyper-alert *prepares for* a later one if the former makes the later easier to be successful
    - Decompose prerequisites and consequences into pieces of predicates
    - Match the predicates

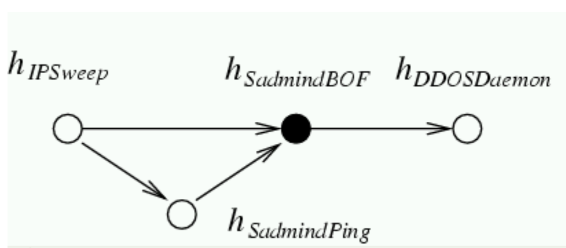


Temporal Constraints

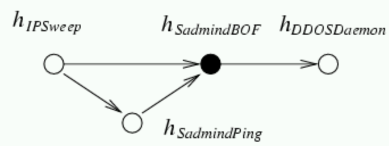
- Definition of hyper-alert
  - Allows alert aggregation,
  - But over flexible: allows alerts in arbitrary time points to be aggregated
- Duration constraint
  - Timestamps of all tuples in the same hyper-alert must be within a certain time period.
- Interval constraint
  - The interval between consecutive tuples (in terms of timestamps) must be less than a given threshold.

Hyper-Alert Correlation Graph

- Hyper-alert Correlation Graph $HG = (N, E)$
  - Directed Acyclic Graph
    - Split hyper-alert if it involves cycles.
  - Nodes: hyper-alerts
  - Edges: $(n_1, n_2) \in E$ iff n_1 prepares for n_2.
    - Transitive edges are omitted for the sake of readability.
  - Intuitive representation of a set of correlated hyper-alerts.

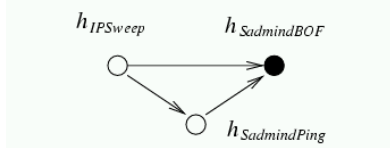


Operations on Hyper-alert Correlation Graphs



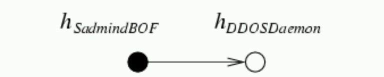
(a) A hyper-alert correlation graph HG

A given graph



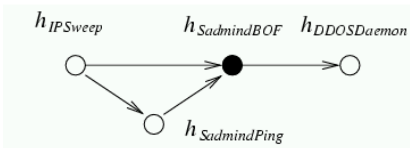
(b) $PG = precedent(h_{SadmindBOF}, HG)$

Precedent operation



(c) $SG = subsequent(h_{SadmindBOF}, HG)$

Subsequent operation

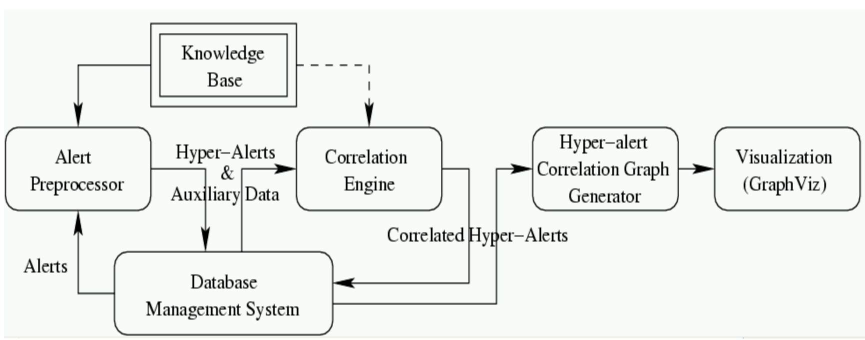


(d) $CG = correlated(h_{SadmindBOF}, HG)$

Correlated operation

Implementation

Architecture of the NCSU Intrusion Alert Correlator



Implementation (Cont'd)

- Preprocessing of alerts
  - *Expanded consequence set*: consequence set + all the predicates implied by the consequence set
  - Encode instantiated predicates as strings
    - Predicate name + "(" + arguments separated by "," +")"
  - Store encoded prerequisite set and expanded consequence sets into two tables (with hyper-alert ID and timestamps):
    - PrereqSet and ExpandedConseqSet.

Implementation (Cont'd)

- Correlation →

```
SELECT DISTINCT c.HyperAlertID, p.HyperAlertID
FROM PrereqSet p, ExpandedConseqSet c
WHERE p.EncodedPredicate = c.EncodedPredicate
        AND c.end_time < p.begin_time
```

Implementation (Cont'd)

- Correctness
  - Assumption 1: Given a set *P* of predicates, for all instantiations of the arguments in *P*, deriving all predicates implied by *P* followed by instantiating all arguments ⇔ instantiating all the arguments and then deriving all the implied predicates.
    - Implication between predicates are true for all attribute values.
  - Assumption 2: All predicates are uniquely identified by names, the special characters "(", ")", and "," do not appear in names and arguments, and the order of arguments in each predicate is fixed.
  - Theorem: *Under assumptions* 1 *and* 2, *our implementation method discovers all and only hyper-alert pairs such that the first one of the pair prepares for the second one.*

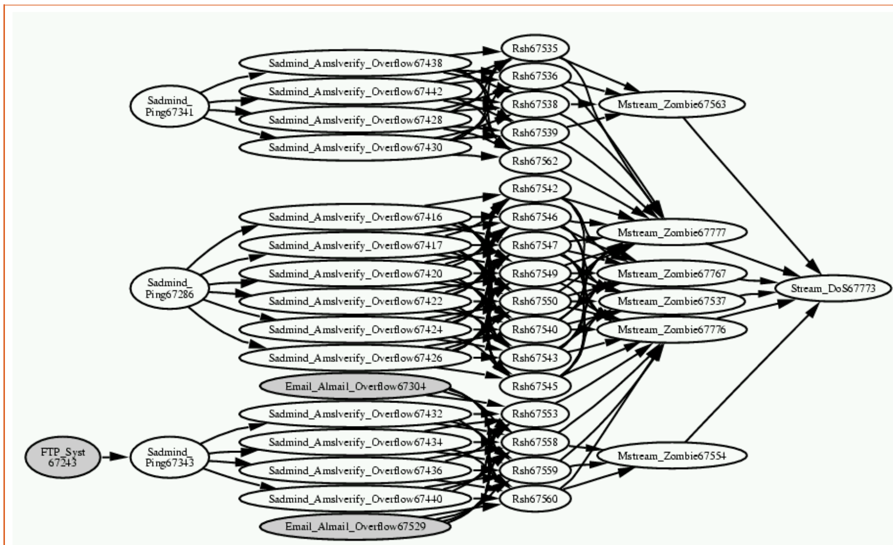
Experimental Evaluation

- Purposes of experiments
  - How well can the proposed method construct attack scenarios?
  - Can alert correlation help differentiate between true and false alerts?

Experimental Evaluation (Cont'd)

- DARPA 2000 intrusion detection scenario specific datasets
  - A novice attacker installs components for and carries out a DDOS attack
  - LLDOS 1.0
  - LLDOS 2.0.2
  - Use NetPoke to replay the network traffic
    - The inside and DMZ traffic of each dataset was replayed separately.
  - Use RealSecure Network Sensor 6.0 to generate alerts
    - Four sets of alerts.

Hyper-Alert Correlation Graph Discovered from the Inside Traffic of LLDOS 1.0



Experimental Evaluation (Cont'd)

- Two measures
  - Completeness: How well can we correlate the related alerts?

$$R_c = \frac{\#\text{Correctly Correlated Alerts}}{\#\text{Related Alerts}}$$

  - Soundness: How correctly are the alert correlated?

$$R_s = \frac{\#\text{Correctly Correlated Alerts}}{\#\text{Correlated Alerts}}$$

Experimental Evaluation (Cont'd)

Completeness and Soundness of Alert Correlation

	LLDOS 1.0		LLDOS 2.0.2	
	DMZ	Inside	DMZ	Inside
# correctly correlated alerts	54	41	5	12
# related alerts	57	44	8	18
# correlated alerts	57	44	5	13
Completeness R_c	94.74%	93.18%	62.5%	66.7%
Soundness R_s	94.74%	93.18%	100%	92.3%

Experimental Evaluation (Cont'd)

Ability to Differentiate Alerts

Dataset		#observable attacks	Tool	#alerts	#detected attacks	Detection rate	#true alerts	False Alert Rate
LLDOS 1.0	DMZ	89	Before	891	51	**57.30%**	57	**93.6%**
			After	57	50	**56.18%**	54	**5.26%**
	inside	60	Before	922	37	**61.67%**	44	**95.23%**
			After	44	36	**60%**	41	**6.82%**
LLDOS 2.0.2	DMZ	7	Before	425	4	**57.14%**	6	**98.59%**
			After	5	3	**42.86%**	3	**40%**
	inside	15	Before	489	12	**80%**	16	**96.73%**
			After	13	10	**66.67%**	10	**23.08%**

√ By attacks we mean attack related actions.

√ Maximum_Coverage policy was used in the experiments. Less aggressive policy would have resulted in smaller false alert rate.