In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, D.C., April 22–24, 2003, Volume I, pages 284–292.

Modeling Multistep Cyber Attacks for Scenario Recognition*

Steven Cheung Ulf Lindqvist Martin W. Fong
System Design Laboratory
SRI International
333 Ravenswood Ave, Menlo Park, CA 94025
{*cheung, ulf, mwfong*}*@sdl.sri.com*

Abstract

Efforts toward automated detection and identification of multistep cyber attack scenarios would benefit significantly from a methodology and language for modeling such scenarios. The Correlated Attack Modeling Language (CAML) uses a modular approach, where a module represents an inference step and modules can be linked together to detect multistep scenarios. CAML is accompanied by a library of predicates, which functions as a vocabulary to describe the properties of system states and events. The concept of attack patterns is introduced to facilitate reuse of generic modules in the attack modeling process. CAML is used in a prototype implementation of a scenario recognition engine that consumes first-level security alerts in real time and produces reports that identify multistep attack scenarios discovered in the alert stream.

1. Introduction

Security alerts are produced mainly by intrusion detection sensors, but also by other sources such as firewalls, file integrity checkers, and availability monitors. A common characteristic for these first-level security alerts is that each isolated alert is based on the observation of activity that corresponds to a single attack step (exploit, probe, or other event). The process of "connecting the dots", that is, correlating alerts from different sensors regarding different events and recognizing complex multistage attack scenarios, is typically manual and ad hoc in nature, and therefore slow and unreliable.

It would be highly desirable to automate the attack scenario recognition process, but there are several challenges facing such efforts:

- Knowledge representing attack scenarios needs to be modeled, preferably in a way that is decoupled from the specifics of a particular correlation technology.
- Producers of first-level security alerts are heterogeneous, and the alert content may vary.
- Attacks belonging to the same scenario could be spatially and temporally distributed.
- First-level alerts could be produced in very high numbers as a result of false positives, repeated probes, or as an attacker-induced smokescreen.
- An attack scenario could be executed in different ways that are equivalent with respect to the attackers' goal. For example, the temporal ordering of some attacks could be changed, or one attack could be substituted for a functionally equivalent one.
- Some attacks that constitute part of a scenario will not be represented in the alert stream. This could be due to missing sensor coverage or because the attack—albeit part of an attack scenario—is indistinguishable from normal benign activity.

In a project called Correlated Attack Modeling (CAM), we have developed methods and a language for modeling multistep attack scenarios, based on typical isolated alerts about attack steps. The purpose is to enable the development of abstract attack models that can be shared among developer groups and used by different alert correlation engines. To verify that the language is suitable for describing attack models to a scenario recognition engine, we have developed such an engine that consumes low-level alerts and makes high-level conclusions based on the scenario models.

The remainder of the paper is organized as follows. Section 2 describes a methodology and requirements for attack modeling. Section 3 presents our modeling language in detail, while the concept of attack patterns is described in Section 4. An implementation of a scenario recognition engine based on our language is described in Section 5. Related

*This work was funded by DARPA through the Air Force Research Laboratory (AFRL) under contract number F30602-00-C-0098.

work is discussed in Section 6, directions for future work are described in Section 7, and conclusions are found in Section 8.

2. Modeling Attacks

Our discussion of attacks and attack steps is guided by the following key definitions:

Vulnerability A condition in a system, or in the procedures affecting the operation of the system, that makes it possible to perform an operation that violates the explicit or implicit security (or survivability) policy of the system

Exploit Single-step (atomic) exploitation of a single vulnerability

Attack step An exploit or other activity performed by an adversary as part of a campaign toward the adversary's goal

(Composite) Attack A collection of one or several attack steps

Attack models for scenario recognition are related to attack trees/graphs used by red teams. However, the purpose of the attack models is not to provide details on how each attack is to be carried out. Instead, the emphasis is on how the attacks are detected and reported. Our modeling methodology includes the following tasks:

- Identify logical attacks in an attack scenario: These attacks may correspond to attack subgoals, and each of them may be further decomposed until it can be detected by a sensor.
- Characterize these logical attacks from the detection point of view: These attacks may be detected by observing certain events, observing certain system states, or performing inferences.
- Specify relationships among these attacks: In particular, there are temporal relationships (e.g., one attack happens before another one), attribute-value relationships (e.g., the target of one attack is the same as the source of another one), and prerequisite relationships (e.g., one attack enables another one to occur).

An attack modeling language must be able to express the knowledge compiled in the modeling tasks described above. In addition, an attack modeling language should fulfill the following requirements to efficiently support attack scenario recognition:

- *Expressive* to cover the range of attacks in which we are interested
- *Unambiguous* to enable mechanization
- Enabling *event reduction* to identify a high-level security event from a large number of low-level incident reports
- Enabling *efficient* implementations
- *Independent* of sensor technologies other than assuming that sensors and correlators use a standard means to communicate

3. Attack Modeling Language

The Correlated Attack Modeling Language (CAML) enables one to specify multistage attack scenarios in a modular fashion. A CAML specification contains a set of *modules*, which specify an inference step. Moreover, the relationships among modules are specified through pre- and postconditions.

Let us consider the following multistep attack scenario as an example: An attacker first exploits a buffer overflow vulnerability of a secure socket layer (SSL) implementation, on which a Web server depends, to obtain remote execution capability. From the Web server, the attacker mounts a file system to access some sensitive data. Then a file corresponding to a Web page is modified to include this data, which the attacker can download using an HTTP request.

The individual attacks of this attack scenario may be observed by different sensors. For instance, a signature-based network IDS may detect the buffer overflow attack step, an anomaly detection component may detect the unusual file access, and a file integrity checker may detect the modification of the Web page. To recognize the "exfiltration" attack scenario, one needs to correlate the pieces.

This scenario has many different variations. For example, instead of using an attack that exploits the SSL vulnerability, the Web server may have other vulnerabilities (e.g., a buffer overflow vulnerability in Windows IIS indexing service [6]) that, when exploited, would enable the remote attacker to run arbitrary code on it. Instead of accessing a file, the attacker may perform network sniffing or query an SQL server to steal data. The large number of different combinations of the attacks makes it difficult to recognize multistage attack scenarios manually. Moreover, explicitly enumerating all these combinations makes attack models less extensible. When a new attack is known, one may have to revisit and modify many previously defined multistage attack models to incorporate it.

In the remainder of this section, we will first illustrate by examples the basic elements of CAML. Then we will describe these elements in detail.

```
1  module OpenSSL-Handshake-BO-2-Remote-Exec (
2  activity:
3      r1: Event(
4          Source(
5              Node(Address(s: address)))
6          Target(
7              Node(Address(t: address))
8              Service(tp: port))
9          Classification(
10             origin == "cve"
11             name == "CAN-2002-0656"))
12 pre:
13     p1: HasService(
14         Node(Address(address == t))
15         Service(
16             imp: implement
17             ver1: version
18             port == tp))
19     p2: Depends(
20         Source(Service(
21             implement == imp
22             version == ver1
23             port == tp))
24         Target(Service(
25             implement == "OpenSSL"
26             ver2: version)))
27     VersionCmp(ver2, "0.9.6") < 0
28     Subset(r1, p1)
29     Subset(r1, p2)
30 post:
31     Event(
32         starttime == r1.starttime
33         endtime == DEFAULT_ENDTIME
34         Source(
35             Node(Address(address == s)))
36         Target(
37             Node(Address(address == t)))
38         Classification(
39             origin == "vendor-specific"
40             name == "CAM-Remote-Exec"))
41 )
```

Figure 1. CAML module: OpenSSL buffer overflow to remote execution

3.1. CAML Examples

Figure 1 shows a CAML module for the SSL buffer overflow attack step. In this example, one may first notice the similarity between the structures of some CAML constructs and the Intrusion Detection Message Exchange Format (IDMEF) [8]. This design facilitates CAML to interoperate with different types of sensors that can generate reports in IDMEF.

In Figure 1, the activity section (cf. Lines 2–11) specifies an event template that could match with intrusion detection reports (which correspond to event instances) for the buffer overflow attack. When a match occurs, the variables in the template (s, t, and tp) are instantiated with the corresponding values in the event instance. For example, s will be instantiated with the source IP address reported by a sensor. Moreover, if a literal is used in the template (e.g., "CAN-2002-0656" as classification name), an event instance must have that value in the corresponding field to match the template.

The pre-condition section (cf. Lines 12–29) specifies the set of conditions that must be met by the event instance and the system state to trigger the inference of this module. The inference results are specified in the post-condition section (cf. Lines 30–40). Specifically, the two predicates — HasService and Depends — and the function VersionCmp()[1] specify that the target host must provide a service associated with port tp that depends on a vulnerable implementation of SSL. Moreover, every event and predicate in CAML is associated with a time interval during which it is valid (start and end time for an event). The temporal predicate Subset() is used to specify that the HasService and the Depends predicates hold when the matching event instance, denoted by the label $r1$, occurs. If the activity and the pre-condition sections are satisfied, the post-condition section says that a remote execution event may result.

The inference result of the OpenSSL-Handshake-BO-2-Remote-Exec module — that is, remote execution — may be used as an input for another module. Figure 2 shows an example of inferring a "data theft" event from a "remote execution" event and an "access violation" event. (The latter may be inferred by other sensor reports such as one corresponding to a suspicious file system mount request.) Another module (not shown here) may correlate a "data theft" event with a "data export" event (which may in turn be inferred from an "integrity violation" event corresponding to an unauthorized Web page modification) to detect the exfiltration scenario. The attack model for this scenario is depicted by an AND/OR tree [27] in Figure 3.

3.2. CAML Modules

A module is the basic unit for specifying correlation steps in CAML. A module specification consists of three sections, namely, *activity*, *pre-condition*, and *post-condition*. To support event-driven inferences, the activity section is used to specify a list of events needed to trigger the module. These events include observed events (corresponding to sensor reports) and inferred events. These events are specified using *event templates*, which describe the requirements for the candidate event instances. The structure of CAML events is based on IDMEF. For instance, the top-level elements of events, similar to those of IDMEF alerts, are Analyzer, Source, Target, Classification, Assessment, and Correlation. CAML also has other fields

[1] The function VersionCmp(a, b) compares two strings a and b. It returns an integer less than, equal to, or greater than zero if the version number a is before, the same as, or after version number b.

```
1  module Remote-Exec-Access-Violation-2-Data-
 Theft (
2  activity:
3      r1: Event(
4          Source(
5              Node(Address(a: address)))
6          Target(
7              Node(Address(b: address)))
8          Classification(
9              origin == "vendor-specific"
10             name == "CAM-Remote-Exec"))
11     r2: Event(
12         Source(
13             Node(Address(address == b)))
14         Target(
15             Node(Address(c: address)))
16         Classification(
17             origin == "vendor-specific"
18             name == "CAM-Access-Violation"))
19 pre:
20     StartsBefore(r1, r2)
21 post:
22     Event(
23         starttime == r1.starttime
24         endtime == r2.endtime
25         Source(
26             Node(Address(address == a)))
27         Target(
28             Node(Address(address == c)))
29         Classification(
30             origin == "vendor-specific"
31             name == "CAM-Data-Theft"))
32 )
```

Figure 2. CAML module: Remote execution and access violation to data theft

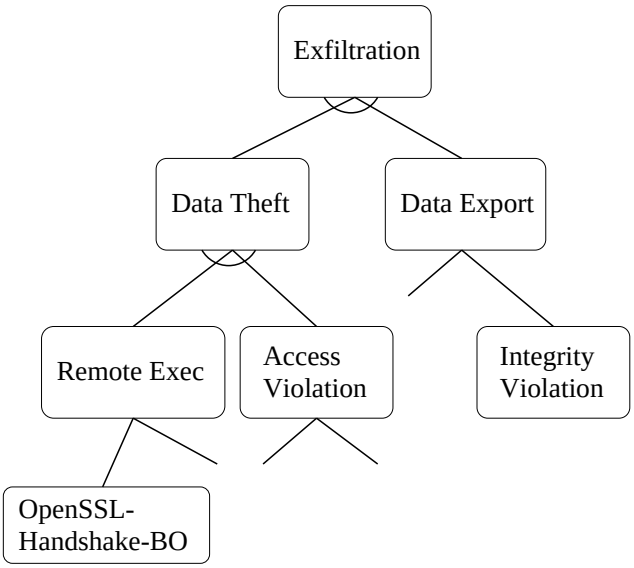

Figure 3. Attack model for the exfiltration scenario

that do not have counterparts in IDMEF. In particular, there are fields for indicating the start time as well as the end time of an attack (for evaluating temporal interval relationships), for reporting alert counts (to facilitate threshold-based analyses), for reporting thread id's (to facilitate alert threading), and for reporting correlated results (to support correlating of correlated results).

Labels may be associated with matching event instances or their fields so that they can be referenced in another part of the module. Moreover, simple constraints in the form of event field comparisons may be used to specify the event sets needed by a module. For example, in Figure 2, the label *b* is used as a handle for the target IP address of a "remote execution" event matched by the first event template. It is subsequently used in a constraint for the source IP address of an event instance matching the second event template. CAML can handle the situations in which an event instance may or may not provide data for a field by means of the *optional field* construct. A label preceded by a "?" means that the field is optional. An event instance does not need to have the fields marked optional to match an event template; however, if it does, these values will be used when the corresponding labels are referenced.

For specifying constraints on the system states and the event instances, predicates may be used in the pre-condition section of a module. (We will discuss these predicates in Section 3.3.) Examples of system state constraints include restrictions on host or service configurations and the state of server integrity. Examples of event constraints that can be specified in this section include temporal interval relationships among events.

If the activity and the pre-condition sections are met, the inference result specified in the post-condition section will hold (in our model). In particular, a module may derive new system states (in the form of predicates) and inferred events. The derived information may then be used to trigger the inference of other modules. As a result, multistep attack scenarios can be detected by chaining the inferences of CAML modules.

3.3. CAML Predicates

To support attack model extension (i.e., ability to incorporate new attack knowledge in attack models) and module composition, we need a uniform way to represent objects and to express their relationships. For the former, CAML uses an IDMEF-based representation for events and predicates. For the latter, we have developed a library of predicates, which functions as a vocabulary to describe the properties of system states and events. Predicate instances may be fed to a correlation engine at startup time; they are specified using a construct called *init section* in CAML. Predicate instances may also be acquired dynamically. In par-

Table 1. CAML predicate categories

Category	Description	Example
Temporal	Relationships between two time intervals, based on Allen's work on temporal intervals [1]	*IsBefore(r1,r2)* indicates time interval *r1* ends strictly before the start of time interval *r2*
Hosts	Properties or states of a host	*SuspiciousHost* indicates suspicious activities originated from a specified host have been observed
Services	Properties or states of an operating system or an application instance	*HasService* indicates a specified host provides a specified service
Files	Relationships and properties pertaining to files	*HasFile* indicates a specified host has a file with a specified name
Users	Relationships and properties pertaining to users	*SwitchUser* indicates a specified user at a specified host can "become" another specified user at another specified host
Know	Specifying the predicate instances known by a particular user	One may indicate a specified user knows the password of another specified user at a specified host

ticular, the post-condition section of a module may specify inferred predicates. Every predicate instance has an associated time interval during which it holds. When information needed to determine the truth value of a predicate is not available, the correlation engine could let the evaluation return a default value based on a policy. For example, the policy could state that missing information should not prevent a module pre-condition from being satisfied, and the default truth values could be calculated accordingly.

Currently, several dozen predicate types have been defined in CAML, and they are divided into six categories, namely, *Temporal*, *Hosts*, *Services*, *Files*, *Users*, and *Know*. The predicate categories are summarized in Table 1.

4. Attack Patterns

Developing attack models for multistep attack scenarios could be quite time-consuming. Moreover, the quality of the models depends heavily on the specifier's experience. Thus, it is important to identify methods for building new attack models based on previously defined ones.

Attack patterns facilitate attack model reuse. These attack patterns correspond to high-level reusable modules that characterize common attack techniques from the detection point of view. The concept of attack patterns is inspired by design patterns [11], which address reuse of software designs and architectures. In particular, software designs that are proven to be effective for solving certain recurring problems in a context are distilled to form design patterns. Similarly, attack patterns are developed to capture the essence of commonly occurring techniques used by attackers. However, the focus for attack patterns is not to facilitate attack development, but to facilitate detection.

A specification of an example attack pattern, called BANDWIDTH AMPLIFIER, is shown in Figure 4. The specification consists of several parts. The *Attack Goal* and the *Considerations* sections provide the context for the attack pattern. The former describes the issue to which the pattern addresses, and the latter discusses the main considerations to determine whether to use this pattern. The *Approach* section describes the attack pattern itself. Known instances of this pattern are described in the *Examples* section. The *Detection* section characterizes this attack pattern from the detection point of view and shows CAML specifications for detecting it. Finally, the *Related Patterns* section describes relationships between this pattern and other attack patterns.

When a new attack is discovered and understood, one may be able to factor the attack into attack patterns. As a result, detecting this attack can be reduced to detecting instances of the attack patterns and their relationships. To illustrate this concept, let us consider two other attack patterns:

COMMANDER-SOLDIER**:** This pattern corresponds to a technique to increase the attack power and to hide the true source of an attack by managing a set of nodes to attack a target. There are two types of components in this pattern, namely, commanders and soldiers. A commander coordinates a number of soldiers to attack the same target, and the soldiers carry out the attack actions.

PERSISTENT INQUIRER**:** This pattern corresponds to a technique that attempts to consume the resources of a node by continuously sending requests to it to prevent it from serving legitimate requests. An example is TCP SYN flooding.

Using these attack patterns, one could model distributed denial of service (DDOS) attacks like mstream [5] and

Pattern Name: Bandwidth Amplifier
Attack Goal: To generate more traffic to jam a target to reduce its availability.
Considerations:
— The communication channel between the source and the target of an attack may have limited bandwidth.
— One may not be able to break-in and use other hosts that have high-bandwidth channels to directly attack the target.
— The break-ins on other hosts could be detected and traced.
— The packets sent to the target should have certain varieties. Otherwise, the defender may be able to detect and to block the attack.
Approach: Using an intermediate node that takes a "small" input and generates a "large" output to flood the target. (The size of the input/output may be measured by the number of bytes or the number of packets.) Moreover, by sending requests with a forged source address, equal to the target's address, to this intermediate, the intermediate will send the responses to the target. As a result, the amount of traffic going to the target can be increased. Because of the anonymous nature of this technique, it is difficult to trace the attack back to its true source.
Examples:

1. Sending DNS requests with a forged source address to cause (large) DNS responses to be sent to the target. The DNS server is the intermediate node. See AusCERT Advisory AL-1999.004 [2].

2. Sending an ICMP echo request whose source address equals the target's address to a broadcast address. In this case, the intermediate node is the network corresponding to this broadcast address. See CERT Advisory CA-1998-01 [3].

Detection: A characteristic of this pattern is that one may observe a large amount of (unsolicited) network traffic going to a node. Moreover, the source(s) of this traffic have the small-input-large-output property. A CAML module for detecting this pattern is as follows:

```
module Packet-Flood-2-Bandwidth-Amplifier (
activity:
    r1: Event(
        Source(
            Node(Address(s: address))
            Service(n: name))
        Target(
            Node(Address(t: address)))
        Classification(
            origin == "vendor-specific"
            name == "CAM-Packet-Flood"))
pre:
    p1: SmallInputLargeOutput(
        Node(Address(address == s))
        Service(name == n))
    Intersects(r1, p1)
post:
    Event(
        starttime == r1.starttime
        endtime == r1.endtime
        Source(
            Node(Address(address == s))
            Service(name == n))
        Target(
            Node(Address(address == t)))
        Classification(
            origin == "vendor-specific"
            name == "CAM-Bandwidth-Amplifier"))
)
```

Related Patterns: Depending on the amplification ratio, the amplified traffic may not be sufficient to jam the target. *Commander-Soldier* may be used with *Bandwidth Amplifier* to consume more network bandwidth of the target.

Figure 4. Attack pattern: Bandwidth Amplifier

TFN [4]. The mstream DDOS attack uses TCP ACK flooding to consume the CPU time of the target. Moreover, multiple machines are used to generate more TCP ACK packets. The mstream attack can be characterized by the attack patterns PERSISTENT INQUIRER and COMMANDER-SOLDIER and their relationships. Another DDOS attack called TFN uses different tactics including TCP SYN flooding and smurf, a DOS attack based on ICMP directed broadcast. A model for TFN may be constructed using three attack patterns, namely, PERSISTENT INQUIRER, BANDWIDTH AMPLIFIER, and COMMANDER-SOLDIER. Depending on which tactic(s) an attacker uses, an instance of TFN may exhibit only behavior pertaining to PERSISTENT INQUIRER or BANDWIDTH AMPLIFIER. Thus, the TFN model should specify an "or" relationship between PERSISTENT INQUIRER and BANDWIDTH AMPLIFIER.

5. Implementing a Scenario Recognition Engine

To validate the practical usefulness of the CAML modeling language, we implemented a scenario recognition engine that uses attack specifications written in CAML. The implementation integrates two advanced technologies developed in the EMERALD program [17, 20], *P-BEST* and *eflowgen*. P-BEST is an expert system shell for building real-time forward-reasoning expert systems based on production rules [15]. Developed as a platform for the EMERALD M-correlator [19], eflowgen is an extensible application framework where application-specific processing modules are triggered in response to events (e.g., messages, timers, and file and network I/O). These processing modules may be dynamically created, modified, reordered, and destroyed. In the scenario recognition engine, eflowgen receives and processes incoming reports and asserts them as facts into the P-BEST factbase. When a fact has been asserted, eflowgen calls the P-BEST inference engine.

5.1. Translating CAML to P-BEST

The first step in building a P-BEST inference engine based on a CAML model is to translate the CAML specification into the P-BEST language. In the described pilot implementation, this translation was performed manually. A CAML module maps fairly well into a P-BEST rule, by letting the activity and pre-condition sections form the antecedent of the rule, while the post-condition section becomes the consequent. Predicates are implemented as facts, each representing a specific predicate, using a generic fact type (P-BEST *ptype*) with a large number of member fields. Each different predicate typically uses a small subset of the available fields. In the traditional version of P-BEST, this would have been unusable, because it required every field of a fact to be populated. We have modified P-BEST to allow sparsely populated facts and added a function that can test if a given field is populated or not. Report events are naturally mapped into P-BEST facts.

5.2. Validation Scenario

In DARPA's Cyber Panel program, a project called the Grand Challenge Problem (GCP) has developed example attack scenarios that can be used for testing and evaluation of detection and correlation technologies. The example mission system in the GCP consists of multiple enclaves with heterogeneous subsystems that are used jointly to perform a mission-critical function.

For the validation of our scenario recognition engine, we chose an attack scenario from GCP version 2.0. The scenario is composed of several coordinated attacks, some of which must occur in a certain temporal order. The resulting CAML specification consists of 14 modules.

The GCP provides alerts in IDMEF (XML) format from intrusion detection sensors enabled at various locations in the example mission system during the attack scenario. The eflowgen component of our scenario recognition engine was instrumented to directly consume the IDMEF alerts and map the information into P-BEST facts.

5.3. Results

The scenario recognition engine could correctly identify the modeled attack scenario from the alert reports. However, the processing latency on a regular desktop computer was in the order of 3 minutes, which is too slow for real-time deployment. Analysis of the runtime behavior of the engine showed that certain parts of the translated CAML code caused combinatorial explosions in P-BEST (nested loops) with pessimistic evaluation. Basically, all expressions were placed in the innermost loop. This problem has not been observed in the extensive use of P-BEST in development of intrusion detection sensors, because such P-BEST code typically has very few fact-matching clauses in rule antecedents. CAML specifications, on the other hand, tend to result in complex antecedents causing this problem to manifest.

We addressed this problem by developing realistic (as opposed to pessimistic) loop evaluation. The P-BEST language was extended with hints that tell the P-BEST compiler on which nesting level a clause with implicit fact references should be placed. We also added explicit syntactical constructs to the P-BEST language for placing selected evaluations outside the nested loops. For example, this can be used for global variables that are independent of facts.

The optimizations resulted in the processing time for the example scenario being reduced from 3 minutes to 1 sec-

ond, which makes it feasible to deploy the scenario recognition engine in real-time situations.

6. Related Work

As observed by Eckmann *et al.*[10], there are several distinct classes of languages that are used to encode different aspects of attacks. In their terminology, CAML and other languages that are used to analyze security alerts belong to the correlation language category.

Alert correlation has been proposed to address the difficult problems of analyzing a large number of alerts (possibly generated by heterogeneous sensors), identifying the security-critical ones and discounting the false alarms, and producing high-level reports to summarize and to explain the alerts. Exemplary intrusion correlation work includes probabilistic correlation based on attribute similarity [26], mission-impact-based correlation that employs common-attribute aggregation, topology analysis, and mission-criticality analysis to perform incident ranking [19], alarm clustering to support root cause analysis [13], IBM Zurich's Tivoli Enterprise Console that employs common-attribute clustering, alert duplication recognition, and event consequence detection [9], Honeywell's Scyllarus correlation framework [12], and a simulated-annealing-based approach for detecting stealthy portscans [24].

For detecting multistep attack scenarios, a naive approach is to use an attack signature that explicitly specifies the constituent attacks and the ordering among them. In fact, this approach is commonly used in signature-based intrusion detection for detecting attacks that involve multiple events. However, extending this approach to recognize (complex) attack scenarios has weaknesses. In particular, because some of the attacks may be substituted by other functionally equivalent ones and the ordering of the attacks could be changed without affecting the outcome, there may be many different variations of an attack scenario. Also, it is difficult to extend these attack signatures to incorporate new attack information.

To address these weaknesses, an attack modeling approach based on specifying the pre-condition and the post-condition of individual attacks has been proposed in Jigsaw [25], LAMBDA [7], ADeLe [16], and by Ning *et al.* [18]. Attacks are related to each other through matching the post-condition of an attack with the pre-condition of another. Moreover, this approach facilitates the specification of functionally equivalent attacks and of new attacks as these attacks can be specified individually. CAML also uses this modular approach for modeling attack scenarios. To support module composition and attack model extension, it is very important to have a uniform interface among modules. This paper differs from prior work in that it focuses on a uniform representation of objects and their relationships and on attack model reuse. We have developed a library of predicates, which functions as a vocabulary to describe system states and events. Developing attack models for multistep scenarios could be time-consuming and complicated. To facilitate reuse of the results of prior modeling efforts, we present a method based on characterizing common attack techniques and using them as higher-level abstractions in attack scenario modeling.

Attack scenario recognition shares many similarities with vulnerability analysis for complex computer and network systems (e.g., [21–23, 28]). In particular, to discover the vulnerabilities of a network, one may need to reason about the configurations of individual hosts, the vulnerabilities of the hosts, and the connectivity and interdependencies among them. Moreover, in vulnerability analysis, characterizing attacks in terms of pre- and post-conditions has also been found useful to infer attack sequences that violate a security policy.

7. Future Directions

We are currently developing extensions to the CAML scenario recognition engine that will introduce dynamic and adaptive functionality. We envision an intelligent component that not only listens to sensors, but actively tunes sensors and other components based on its global view of the cyber battlefield. For example, in a situation of increased threat level, it could invoke additional analysis engines, trigger response components, or update configurations of sensors to extend their monitoring or make sensors switch into an active probing mode [14]. Furthermore, the engine will be able to dynamically accept new or updated CAML models and update its knowledge base with those models during runtime.

8. Conclusions

We have presented methods and a language, called CAML, for modeling multistep attack scenarios in a way that enables correlation engines to use these models to recognize attack scenarios. CAML uses a modular approach for specifying attack scenarios, making the models expressive and extensible. Each module represents an inference step, and these modules can be linked together to recognize attack scenarios. To facilitate module interfacing, CAML has a set of predicates for specifying the properties of system states and events and employs a uniform representation for events and predicates. A concept called *attack patterns* facilitates reuse of modules and attack models. To validate our approach, we implemented a prototype scenario recognition engine that uses CAML specifications to identify an attack scenario in a stream of IDMEF-encoded alerts.

Acknowledgments

The authors are grateful to the following colleagues who have made valuable contributions to the CAM project: David Farrell, John Khouri, Phillip Porras, Sami Saydjari, Rico Valdez, and Bradley Wood. We also thank Catherine McCollum, DARPA, for her support and encouragement.

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] Australian Computer Emergency Response Team. *Denial of Service (DoS) attacks using the Domain Name System (DNS)*, Aug. 13, 1999. AusCERT Advisory AL-1999.004.

[3] CERT Coordination Center. *Smurf IP Denial-of-Service Attacks*, Jan. 5, 1998. CERT Advisory CA-1998-01.

[4] CERT Coordination Center. *Distributed Denial of Service Tools*, Nov. 18, 1999. CERT Incident Note IN-99-07.

[5] CERT Coordination Center. *"mstream" Distributed Denial of Service Tool*, May 2, 2000. CERT Incident Note IN-2000-05.

[6] CERT Coordination Center. *Buffer Overflow In IIS Indexing Service DLL*, June 17, 2001. CERT Advisory CA-2001-13.

[7] F. Cuppens and R. Ortalo. LAMBDA: A language to model a database for detection of attacks. In H. Debar, L. Mé, and S. F. Wu, editors, *Recent Advances in Intrusion Detection (RAID 2000)*, volume 1907 of *LNCS*, pages 197–216, Toulouse, France, Oct. 2–4, 2000.

[8] D. Curry and H. Debar. *Intrusion Detection Message Exchange Format: Data Model and Extensible Markup Language (XML) Document Type Definition*. Intrusion Detection Working Group, June 20, 2002. Work in progress, IETF Internet-Draft draft-ietf-idwg-idmef-xml-07.txt.

[9] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In W. Lee, L. Mé, and A. Wespi, editors, *Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 85–103, Davis, California, Oct. 10–12, 2001.

[10] S. T. Eckmann, G. Vigna, and R. A. Kemmerer. STATL: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10:71–103, 2002.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[12] R. P. Goldman, W. Heimerdinger, S. A. Harp, C. W. Geib, V. Thomas, and R. L. Carter. Information modeling for intrusion report aggregation. In *DARPA Information Survivability Conference and Exposition (DISCEX II)*, volume 1, pages 329–342, Anaheim, California, June 12–14, 2001.

[13] K. Julisch. Mining alarm clusters to improve alarm handling efficiency. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, pages 12–21, New Orleans, Louisiana, Dec. 10–14, 2001.

[14] U. Lindqvist. The inquisitive sensor: A tactical tool for system survivability. In *Supplement of the 2001 International Conference on Dependable Systems and Networks*, pages C–14–C–16, Göteborg, Sweden, July 1–4, 2001.

[15] U. Lindqvist and P. A. Porras. Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 146–161, Oakland, California, May 9–12, 1999.

[16] C. Michel and L. Mé. ADeLe: An attack description language for knowledge-based intrusion detection. In M. Dupuy and P. Paradinas, editors, *Trusted Information: The New Decade Challenge: IFIP TC11 16th International Conference on Information Security (IFIP/SEC'01)*, pages 353–368, Paris, France, June 11–13, 2001.

[17] P. G. Neumann and P. A. Porras. Experience with EMERALD to date. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, California, Apr. 9–12, 1999.

[18] P. Ning, S. Jajodia, and X. S. Wang. Abstraction-based intrusion detection in distributed environments. *ACM Transactions on Information and System Security*, 4(4):407–452, Nov. 2001.

[19] P. A. Porras, M. W. Fong, and A. Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In A. Wespi, G. Vigna, and L. Deri, editors, *Recent Advances in Intrusion Detection (RAID 2002)*, volume 2516 of *LNCS*, pages 95–114, Zurich, Switzerland, Oct. 16–18, 2002.

[20] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, Baltimore, Maryland, Oct. 7–10, 1997.

[21] C. R. Ramakrishnan and R. Sekar. Model-based analysis of configuration vulnerabilities. *Journal of Computer Security*, 10(1,2):189–209, 2002.

[22] R. W. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 156–165, Oakland, California, May 14–17, 2000.

[23] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 273–284, Oakland, California, May 12–15, 2002.

[24] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10:105–136, 2002.

[25] S. J. Templeton and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of the 2000 New Security Paradigms Workshop*, pages 31–38, Ballycotton Co., Cork, Ireland, Sept. 18–21, 2000.

[26] A. Valdes and K. Skinner. Probabilistic alert correlation. In W. Lee, L. Mé, and A. Wespi, editors, *Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 54–68, Davis, California, Oct. 10–12, 2001.

[27] P. H. Winston. *Artificial Intelligence*. Addison-Wesley, 1977.

[28] D. Zerkle and K. Levitt. NetKuang–a multi-host configuration vulnerability checker. In *Proceedings of the Sixth USENIX Security Symposium*, pages 195–204, San Jose, California, July 22–25, 1996.