EVSec: An Approach to Extract and Visualize Security Scenarios from System Logs

Jameleddine Hassine
jhassine@kfupm.edu.sa
Information and Computer Science Department, KFUPM
Interdisciplinary Research Center for Intelligent Secure Systems, KFUPM
Dhahran, KSA

ABSTRACT

Logs, a.k.a. execution traces, provide a glimpse into the functionalities of running systems that have poor, incomplete, or outdated documentation. Logs contain a rich amount of information that can be used to facilitate troubleshooting/debugging, track events, detect security breaches, maintain regulatory requirements, and profile user behavior and workload. Driven by the growing complexity of today's software platforms, reverse engineering of high-level models from system logs has gained momentum in recent years. In this paper, we introduce *EVSec*, an approach to extract and visualize security scenarios from system logs. The collected logs are first merged, filtered, labeled, and segmented into *execution phases*. The resulting phases are then visualized using the ITU-T standard, Use Case Maps (UCM) notation, extended with security annotations. We show the applicability of our proposed *EVSec* approach using two real-world security features, namely, Cisco IOS Login block and Cisco Unicast Reverse Path Forwarding (uRPF).

CCS CONCEPTS

• **Security and privacy → Software reverse engineering**; • **Human-centered computing → Information visualization**.

KEYWORDS

Logs, extraction, filtering, visualization, security scenarios, Use Case Maps (UCM), Cisco security features

ACM Reference Format:
Jameleddine Hassine. 2022. EVSec: An Approach to Extract and Visualize Security Scenarios from System Logs. In *The International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE 2022), June 13–15, 2022, Gothenburg, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3530019.3531338

1 INTRODUCTION

Security threats are causing considerable financial losses for governments, organizations, and individuals. According to the 2020 IBM data security breach report [11], that surveyed 524 organizations from 17 industries and located in 17 countries/regions, the average time required to identify a security breach is 207 days. Security features have been introduced to help protect systems against malicious attacks. Security features, generally implement a logging system that enables an organization to record important events and to detect security incidents. Typically, a system log is a sequence of temporal events captured during a particular execution of a system. Each log entry can contain user activities, events triggered during execution, and software execution paths. Implementing proper log management and monitoring will ensure a timely detection of an incident or a security breach.

Dynamic analysis aims to analyze system behavior at run time [1]. It does not require the availability of source code and make use of system logs instead. Dynamic analysis, nevertheless, suffers from the size explosion problem [17]. Many log abstraction techniques have been proposed to address this issue [14], which allow for the grouping of execution points that share certain attributes, resulting in a more abstract representation.

The widespread interest in dynamic analysis of software quality attributes provides the major motivation of this research. We, in particular, focus on extracting and visualizing security scenarios from system logs implementing security features. This paper provides the following contributions:

- It introduces the *EVSec* approach to recover and visualize security scenarios from native system logs. *EVSec* is intended to be used primarily by analysts who want to monitor the triggering of security features, without having to look into the distributed and cumbersome logs. Collected logs are merged, filtered, and segmented into clusters, called "execution phases", describing the executed security scenario. The segmented log is visualized as a hierarchy of UCM (Use Case Maps [12]) maps, extended with security annotations [9] and implemented within the jUCMNav [13] tool. Use Case Maps allow for the visual integration of multiple scenarios from many architectural components into one single map. Furthermore, the use of UCM stub/plugin concept to hide the details of execution phases enhances the readability of the produced UCM and provides a scalable solution for the visualization problem.
- It demonstrates the applicability of the *EVSec* approach by applying it to two real-world network security features, namely, Cisco IOS Login block and Cisco Unicast Reverse Path Forwarding (uRPF).

The remainder of this paper is organized as follows. Related work is presented in Sect. 2, followed by the necessary research background in Sect. 3. Section 4 describes the details of the proposed *EVSec* approach. The experimental evaluation of *EVSec* is

EASE 2022, June 13–15, 2022, Gothenburg, Sweden
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9613-4/22/06…$15.00
https://doi.org/10.1145/3530019.3531338

presented in Sect. 5. Section 6 discusses the threats to validity of our proposed approach. Finally, conclusions are drawn in Sect. 7.

2 RELATED WORK

Log security analysis has recently received much attention due to its importance in identifying and potentially stopping intruders attacks. Many strategies to analyze security logs have been proposed in the literature [6, 15, 16, 18], including machine learning, anomaly detection, text analysis, data mining, event correlation, statistical analysis, etc. These techniques can be discussed from log abstraction, log analysis, and log visualization perspectives. Given the limited space, we will focus on the visualization aspects in describing security from log data, in order to alleviate the overhead that analysts face when dealing with high volume of logs.

In [16], visualizations for security analysis were categorized into hierarchical and non-hierarchical. Zhang et al. [18] presented a survey of security visual analytics, where the authors identified five visualization classes, namely, text-based, parallel, hierarchical, three-dimensional (3D), and other designs. Haggerty and Hughes-Roberts [6] proposed an approach for the visualization of log files to aid security post-incident analysis. Two interactive visualization schema were introduced: (1) a network graph, showing events or IP addresses as network nodes and lines to represent relationships between them, (2) a tag cloud, created from the log text, to produce a narrative view of qualitative data (according to the frequency of occurrence, which will reduce noise and highlight important aspects).

In this paper, we propose to use the Use Case Maps (UCM) [12] notation to visualize security system logs. UCMs have been successfully used to visualize availability requirements from execution traces [8]. The most closely related work to ours is by Hassine et al. [10], who introduced a dynamic analysis framework to recover high availability (HA) scenarios from system execution traces. The authors [10] opted for the UCM language [12] in order to visualize the retrieved HA scenarios. In this paper, we follow a similar approach in extracting and visualizing security scenarios from system logs.

3 RESEARCH BACKGROUND

In an attempt to make this paper self-contained, we include some of the core background information relevant to this research. We start by providing a brief description of the security tactics, introduced by Bass et al. [2], then we present the Use Case Maps (UCM) [12] notation and its security annotations [9].

3.1 Security tactics

Bass et al. [2] introduced a comprehensive classification of security tactics based on whether they handle attacks detection, resistance, reaction or recovery. Four classes have been introduced and refined into a number of tactics: (1) Detecting attacks (e.g., intrusions, service denial, compromised integrity, and man-in-the-middle), (2) reacting to attacks (e.g., revoke access, lock computer and inform actors), (3) recovering from attacks (e.g., trace and identify the attacker and restore the service using availability tactics, such as redundancy) are aimed at dealing with successful attacks, (4) resisting to attacks (e.g., Limit access, Limit exposure, etc.), aiming to protect a system's confidentiality and integrity.

3.2 The Use Case Maps (UCM) Notation

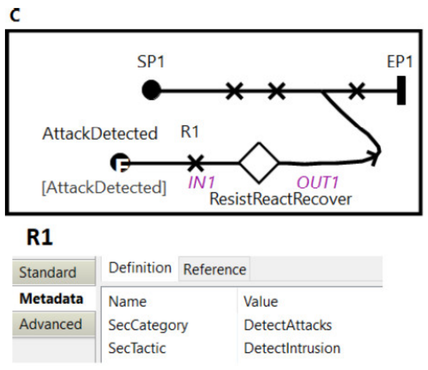
The Use Case Maps (UCM), part of the ITU-T User Requirements Notation (URN) [12] standard, provides a visual and abstract description of scenarios in terms of causal relationships between responsibilities (✕, i.e., steps of a scenario) along paths (a map-like) allocated to a set of architectural components, represented as ▭). A UCM normal path starts with one or many start points (●) and ends with one or many end points (▮). UCM scenarios can be integrated sequentially, as alternatives (with OR-forks —< and OR-joins >—), or concurrently (with AND-forks —⊢ and AND-joins ⊣—). When maps become large to be represented as one single UCM, their details can be hidden in sub-diagrams, called *plug-in* maps, contained in stubs (◇). A plug-in map is bound (i.e., connected) to its parent map by binding the in-paths of the stub with the start points of the plug-in map and by binding the out-paths of the stub to end points of the plug-in map.

UCM allows for the definition of exception paths that start with a failure start point (Ⓕ) and has a guard condition that can be initialized as part of a scenario definition (i.e., scenario triggering condition) or can be modified as part of a responsibility expression. The URN standard [12] supports the *Metadata* mechanism allowing the profiling of the notation to a particular domain. URN *Metadata* describes name-value pairs (i.e., a name (string) and a value (string)) that can be used to tag any URN specification or its model elements, hence providing an extensible semantics to URN. *Metadata* feature is supported by *jUCMNav* [13], the most comprehensive URN tool available to date. For a complete description of the Use Case Maps language, interested readers are referred to the ITU-T standard [12].

3.3 UCM Security annotations

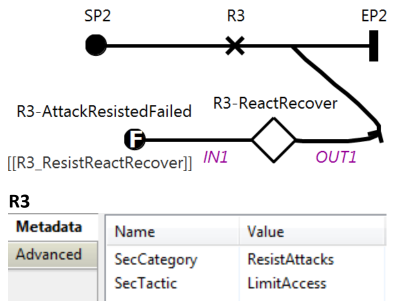
Security tactics [2] have been used as a basis for extending the UCM language [12] with security annotations [9]. Security requirements are modeled at the responsibility and at the scenario path levels. At the *Responsibility* level, the following two metadata attributes may be attached to a responsibility: (1) **SecCategory** denoting the tactic category, if any, that the responsibility is implementing, and (2) **SecTactic** denoting the deployed security tactic. A detailed definition of these attributes and their possible values is described in the UCM security metamodel, introduced in [9]. The assumption of having a responsibility realizing one single tactic has been relaxed in this research, to allow a responsibility to implement many tactics. At the scenario path level a hierarchical structure of cascading failure scenario paths can be used to model different categories of security tactics (i.e., detection, resistance, reaction, recovery).

Figure 1(a) illustrates an example of a UCM security scenario, where the successful detection of an attack (modeled at the normal scenario path using responsibility "*R1*" that implements the *DetectIntrusion* tactic) triggers a failure scenario path, by setting the failure guard *R1-AttackDetected* to true. Consequently, the system may be able to resist the ongoing attack using responsibility "*R3*" in Fig. 1(b) (part of the plugin map enclosed in the "*R1-ResistReactRecover*" static stub of Fig. 1(a)) that implements the *LimitAccess* tactic. In



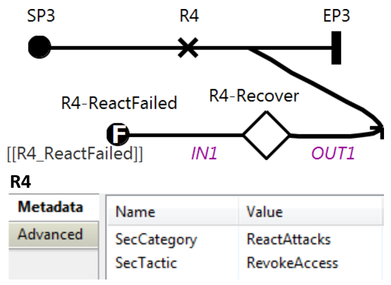
Name	Value
SecCategory	DetectAttacks
SecTactic	DetectIntrusion

(a) UCM Security scenario root map



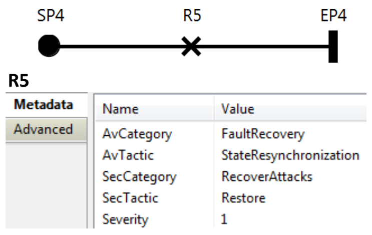
Name	Value
SecCategory	ResistAttacks
SecTactic	LimitAccess

(b) R1-ResistReactRecover plugin



Name	Value
SecCategory	ReactAttacks
SecTactic	RevokeAccess

(c) R3-ReactRecover plugin



Name	Value
AvCategory	FaultRecovery
AvTactic	StateResynchronization
SecCategory	RecoverAttacks
SecTactic	Restore
Severity	1

(d) R4-Recover plugin

Figure 1: UCM Modeling of attack detection, resistance, reaction, and recovery

the case of an unsuccessful resistance, a system may be able to react to the attack using responsibility "*R4*" in Fig. 1(c) (part of the plugin map enclosed in the "*R3-ReactRecover*" static stub in Fig. 1(b)) that implements the *RevokeAccess* tactic). Finally, the system may be compromised (e.g., lost data, etc.). In such a case, the system should be able to recover from the attack using responsibility "*R5*" in Fig. 1(d) (part of the plugin map enclosed in the "*R4-Recover*" static stub in Fig. 1(c)) that implements the *Restore* tactic, which is refined using the availability tactics. For a description of the UCM-based availability tactics, interested readers are referred to [7].

It is worth noting that upon a system might not implement the four categories of tactics, which will reduce the number of levels in the presented cascading stub hierarchy.

4 EVSEC: EXTRACTION AND VISUALIZATION OF SECURITY SCENARIOS

In this section, we describe our proposed approach to extract and visualize security scenarios. The approach is composed of four main phases, as shown in Fig. 2: (1) Log merging, (2) Log filtering, (3) Log tagging and segmenting, and (4) Security scenario visualization.

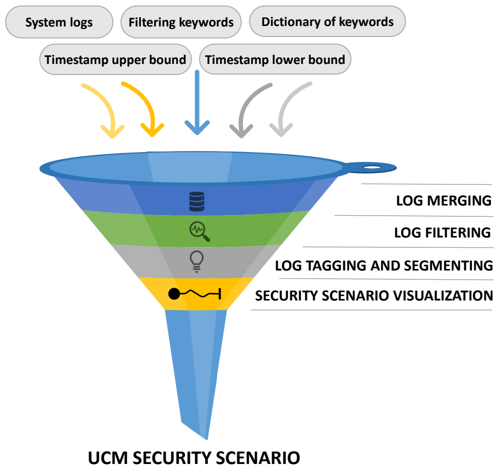


Figure 2: Security requirements recovery approach

4.1 Log merging

The first step consists of merging all system logs from all components relevant to the conducted analysis. Although, the content and format of logs can vary from one manufacturer to another and even among systems from the same manufacturer, log entries share many common attributes such as timestamps, the process ID generating the event/error, operation/event prefix, severity of the event, and a brief description of the event/error. In our context, the log merging is performed according to the events' timestamps. All log entries from all components are ordered chronologically. Furthermore, since the log size is huge, the analyst may specify a log window, i.e., lower and upper bound timestamps. The output of this step is a consolidated log that preserves all log entries as well as their originating components.

4.2 Log filtering

Since not all the captured data is useful for our security analysis task, the second step consists of filtering the merged system logs, through applying customization criteria to include or exclude log entries based on features/protocols names, types of specific systems or services (e.g., IP address), administrator operator actions (e.g., add/remove configurations, shut/unshut network interfaces, etc.), events to monitor (e.g., session timeout, network interfaces state changes, neighbors up/down, etc.). A list of filtering keywords may be supplied as input to this step. The output of this step is a merged and reduced log.

4.3 Log tagging and segmenting

Once the log is filtered, we proceed with tagging individual log entries as "User" (i.e., entries relative to user actions), "Sec" (i.e., entries describing security events/actions, such as login block, denial of service), and "Network" (i.e., entries describing non-security system events; these are general network events). The tagging is performed according to a dictionary of user-defined keywords. Given the variety of network features/events, an entry that cannot be tagged as "User" or "Sec" is considered as part of "Network" category.

The labeled log entries are then parsed sequentially and grouped into "execution phases". An execution phase is composed of a sequence of log entries tagged with the same label, e.g., "User", "Sec", or "Network". A new execution phase is created when we come by a log entry with a different tag. The output of this step is a segmented and tagged log.

4.4 Security scenario visualization

The fourth step consists of mapping the "execution phases" into Use Case Maps (UCM) models with security annotations. Each log entry is mapped to a single UCM responsibility. To increase the readability and the scalability of the resulting UCM, an execution phase with more than three responsibilities is presented using a static stub, that encloses a plugin map. "User" or "Network" execution phases should be placed in the normal path, while "Sec" execution phases should be placed in an exception path (i.e., a scenario path starting with a failure start point F) attached to the normal path using a UCM Or-Join.

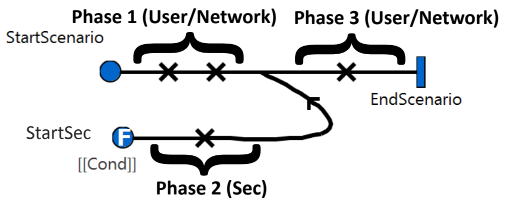


Figure 3: Generic Security Requirements Visualization

A generic UCM security scenario is shown in Fig. 3. By enabling the triggering condition within the final responsibility before the Or-Join (e.g., *"Cond := true"*), control is transferred from the normal path to the exception path. Despite the fact that this triggering condition is not part of the execution phase, it is required in order to maintain phase causality and ensure that the final model adheres to UCM dynamic traversal semantics. The assignment of a specific security tactic category/type to a responsibility is performed based on the existing dictionary of keywords.

It is worth noting that the execution phases/responsibilities may be part of different UCM components (not shown in Fig. 3).

5 EXPERIMENTAL EVALUATION

In this section, we apply our proposed approach to two Cisco IOS security features: (1) Cisco IOS Login Block feature, and (2) Cisco Unicast Reverse Path Forwarding (uRPF) feature.

5.1 Mapping Cisco IOS log entries to jUCMNav metadata attributes

Cisco IOS execution log messages [3] can contain up to 80 characters and have the following format:

```
timestamp: %facility-severity-MNEMONIC:description
```

where the *timestamp* is followed by *facility* denoting the system or the device with logging enabled, e.g., PARSER (parser), SEC (IP Security). *Severity*, between 0 and 7, represents the severity of the message. *MNEMONIC* uniquely describes the message and *description* contains detailed information about the logged event.

The produced Cisco IOS log from both case studies is mapped to jUCMNav as follows: Each log entry is mapped to one responsibility (i.e., responsibility name, description, and metadata attributes). All responsibility fields follow a one-to-one mapping schema, except for the responsibility's *Name*, which is limited to the first *N* characters (N may be supplied as a parameter) of the description. An excerpt of the description can be used in order to improve the readability of the UCM. However, sometimes descriptions are difficult to shorten and still maintain accuracy and readability. For this reason, the full description of the log entry is stored in the *Description* field. Table 1 summarizes the mapping.

Table 1: Mapping of Cisco IOS log fields to jUCMNav responsibility metadata

jUCMNav responsibility metadata attribute	**Cisco IOS log entry field**
Timestamp metadata attribute	timestamp
Facility metadata attribute	facility
Severity metadata attribute	severity
Mnemonic metadata attribute	MNEMONIC
Description	description
Name (first *N* characters or an excerpt of the description)	description

5.2 Experimental setup

We have used GNS3 (Graphical Network Simulator) [5] as our simulation tool. GNS3 is a graphical, open source software that allows for the emulation of complex networks in a virtual environment.

Figure 4 illustrates our testbed topology. In our setup, we use four Cisco c7200 routers (*Hacker*, *R2*, *R3*, and *Target*) running Cisco IOS 15.2(4)S6 software release. OSPF (Open Shortest Path First) routing protocol is provisioned on routers *Target* (on interfaces pos1/0 and pos3/0), *R2* (on interfaces ethernet4/0 and pos1/0), *R3* (on interfaces ethernet4/1 and pos3/0), and *Hacker* (on interface ethernet4/1 only). Hence, the loopback address of *Hacker* (i.e., 1.1.1.1/32) is not advertised on interface ethernet4/0 since OSPF is disabled on that interface. Instead, a default route is configured on *Hacker* in order to route traffic through ethernet4/0 in case of OSPF failure.

It is worth noting that we don't have access to the router *Hacker* since it denotes the malicious actor.

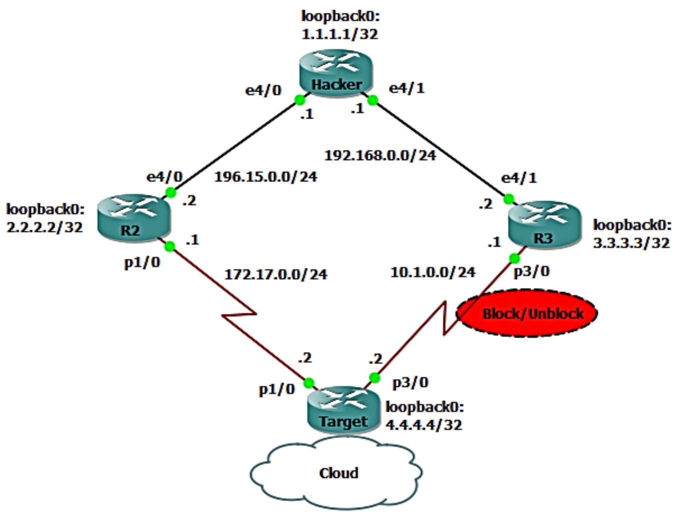


Figure 4: Experimental testbed

5.3 Case study 1: Cisco IOS Login Block

The management of a router, at either the user or executive level, is most frequently performed using Telnet or SSH (secure shell) from a remote host. If the connection address of a router is discovered and is reachable, two types of attacks can be organized against it:

- A malicious user may attempt to interfere with the normal operations of the router by flooding it with connection requests, referred to as *Denial-of-Service (DoS)* attack. The router may become too busy trying to process the repeated login connection attempts to properly handle normal routing services or may not be able to provide the normal login service to legitimate system administrators.
- Unlike a typical DoS attack, a malicious user may attempt a dictionary attack in order to gain administrative access to the router. A dictionary attack is an automated process to try to login by attempting thousands, or even millions, of username/password combinations. The profile for such attempts is typically the same as for DoS attempts; multiple login-attempts in a short period of time.

The Cisco IOS Login Block feature, also called *Login Enhancements*, allows users to enhance the security of a router by configuring options to automatically block repeated failed login attempts by refusing further connection requests (login blocking). Furthermore, it can slow down “dictionary attacks” by enforcing a “quiet period”, to protect the router from DoS attacks. Legitimate connection attempts can still be permitted during a quiet period by configuring an Access Control List (ACL) with the addresses that you know to be associated with system administrators.

Security feature configuration: We have configured Login Block feature on router *Target* to enter a 100 second quiet period if more than 5 login failures occur in 100 seconds or less. All login requests are denied during the quiet period. A login delay of 2 seconds is applied between two login attempts. All successful/failed logins are logged. The used Login Block configuration is as follows:

```
login block-for 100 attempts 5 within 100
login delay 2
login on-failure log
login on-success log
```

Attack scenario: In our attack scenario, we attempt to telnet into router *Target* from router *Hacker*. The authentication of the first five login attempts failed because of invalid user/password, causing the router to enter the quiet mode. Once the block period expires, we issue another login attempt from router *R3* using valid administrator credentials.

Log merging, filtering, and segmenting: Console logs collected from routers *R2*, *R3*, and *Target* are merged then filtered. Only the log from router *Target* is relevant for our attack scenario, as no activity were observed on routers *R2* and *R3*. Fig. 5 illustrates the filtered log. Following the guidelines introduced in Sect. 4, the filtered log has been segmented into three phases. The first phase described the feature configuration entered by the user. The second phase, labeled as “Sec” denotes all security related log entries. Finally, Phase3 shows the user *admin* logout from the router.

UCM Scenario visualization: Figure 6(a) illustrates the resulting UCM of the log presented in Fig. 5. Each log entry is mapped to a responsibility and stored using jUCMNav metadata according to the schema introduced in Sec. 5.1. Since phases 1 and 3 have more than three responsibilities each, their corresponding UCM maps, i.e., Fig. 6(b) and Fig. 6(c), have been refactored into two static stubs “*userConsoleLoggedCommand*” and “*LoginBlock*”. The Boolean condition *C1* has been added to the failure point to ensure the transfer of control from the stub “*userConsoleLoggedCommand*” to the security scenario path.

The first five responsibilities of the “LoginBlock” stub plugin (Fig. 6(c)) corresponding to the first five login failures, implement the *IdentifyActors*, *AuthenticateActors*, and *AuthorizeActors* tactics. Responsibility “*QuiteModeOn*” implements the *LockComputer* tactic.

5.4 Case study 2: Cisco Unicast Reverse Path Forwarding (uRPF)

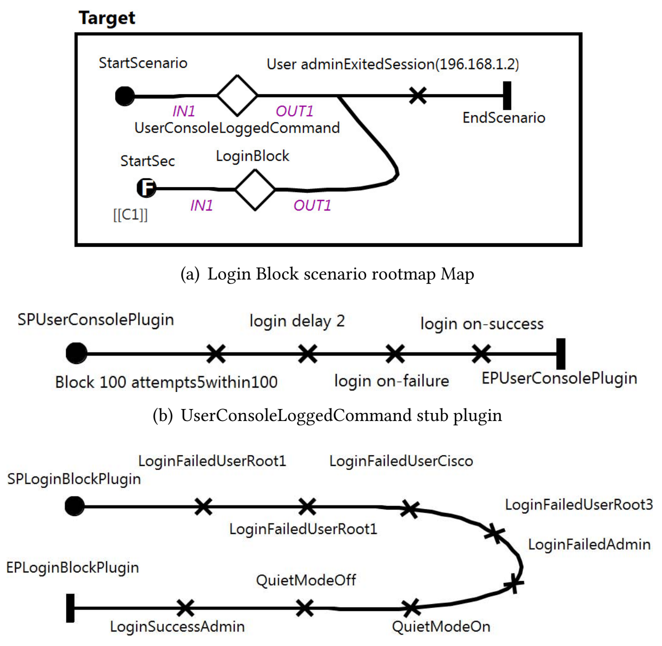
Unicast Reverse Path Forwarding (uRPF) [4] is used to limit malicious traffic on an enterprise network by verifying the reachability of the source address in packets being forwarded; hence limiting the appearance of spoofed addresses on a network. If the source IP address of a packet is not valid, the packet is discarded. There are two *uRPF* modes: (1) Strict mode: Checks whether the source address of the received packet exists in the routing table and that the source address is reachable by a path through the input interface (the interface on which the packet enters the router), and (2) Loose mode: Checks whether the source address of the received packet exists in the routing table and that a valid path through any interface.

Security feature configuration: We have configured strict uRPF on interface POS1/0 on router *Target*. In order to avoid silent drops of packets, a common practice is to configure an ACL (Access Control List) to capture traffic denial. The used uRPF configuration is as follows:

```
interface POS1/0
ip address 172.17.0.2 255.255.255.0
ip verify unicast source reachable-via rx 100
ip access-list extended 100
```

Component	Merged and filtered log	Execution Phase	Execution Label
Target	*Jun 8 16:43:07.015: %PARSER-5-CFGLOG_LOGGEDCMD: User:console logged command:login block-for 100 attempts 5 within 100 *Jun 8 16:43:07.019: %PARSER-5-CFGLOG_LOGGEDCMD: User:console logged command:login delay 2 *Jun 8 16:43:07.019: %PARSER-5-CFGLOG_LOGGEDCMD: User:console logged command:login on-failure log *Jun 8 16:43:07.895: %PARSER-5-CFGLOG_LOGGEDCMD: User:console logged command:login on-success log	1	User
Target	*Jun 8 16:43:51.327: %SEC_LOGIN-4-LOGIN_FAILED: Login failed [user: root] [Source: UNKNOWN] [localport: 23] [Reason: Login Authentication Failed] *Jun 8 16:44:02.319: %SEC_LOGIN-4-LOGIN_FAILED: Login failed [user: root] [Source: UNKNOWN] [localport: 23] [Reason: Login Authentication Failed] *Jun 8 16:44:12.803: %SEC_LOGIN-4-LOGIN_FAILED: Login failed [user: cisco] [Source: UNKNOWN] [localport: 23] [Reason: Login Authentication Failed] *Jun 8 16:44:33.223: %SEC_LOGIN-4-LOGIN_FAILED: Login failed [user: root] [Source: UNKNOWN] [localport: 23] [Reason: Login Authentication Failed] *Jun 8 16:44:44.471: %SEC_LOGIN-4-LOGIN_FAILED: Login failed [user: admin] [Source: UNKNOWN] [localport: 23] [Reason: Login Authentication Failed] *Jun 8 16:44:44.475: %SEC_LOGIN-1-QUIET_MODE_ON: Still timeleft for watching failures is 2 secs, [user: admin] [Source: UNKNOWN] [localport: 23] [Reason: Login Authentication Failed] [ACL: sl_def_acl] *Jun 8 16:46:24.471: %SEC_LOGIN-5-QUIET_MODE_OFF: Quiet Mode is OFF, because block period timed out *Jun 8 16:46:40.551: %SEC_LOGIN-5-LOGIN_SUCCESS: Login Success [user: admin] [Source: UNKNOWN] [localport: 23]	2	Sec
Target	*Jun 8 16:48:13.827: %SYS-6-LOGOUT: User admin has exited tty session 2(10.1.0.1)	3	User

Figure 5: Log relative to the login block feature



(a) Login Block scenario rootmap Map

(b) UserConsoleLoggedCommand stub plugin

(c) LoginBlock stub plugin

Figure 6: Login Block log visualization

```
10 deny ip any any log
```

Attack scenario: In our attack scenario, we start an extended ping from router *Hacker* towards router *Target* using the source IP address of the Hacker (i.e., 1.1.1.1). The ping packets travel through the interface e4/1 on *Hacker*, using the OSPF adjacencies, to reach router *Target* on interface POS3/0 (note that POS3/0 is not protected). Next, we shut down the interface POS3/0 on *Target* forcing the ping traffic to take the e4/0 path on router *Hacker* (since OSPF adjacency with R3 is down). The ICMP Echo Request packets reach router *Target* on interface POS1/0, configured with uRPF. Since the router *Hacker* IP address 1.1.1.1 is not reachable through this interface, these packets will be denied. Once the interface POS3/0 is unshut, ping traffic will flow again through POS3/0 on router *Target*.

Log merging, filtering, and segmenting: Console logs collected from routers *R2*, *R3*, and *Target* are merged then filtered. Similar to the Login Block feature, only the log from router *Target* is relevant for our attack scenario, as no activity were observed on routers *R2* and *R3*. The console log collected and filtered from router *Target* is shown in Fig. 7. Next, the filtered log has been segmented into five phases. The first two user actions are grouped as phase 1. The network events caused by the user configuration are grouped as phase 2. The log entry showing the denial of ICMP packets represents phase 3, which is labeled as “Sec”.

UCM Scenario visualization: Figure 8 illustrates the resulting UCM of the segmented log of Fig. 7. Responsibility “*List 100 denied icmp 1.1.1.1*$\longrightarrow$ *4.4.4.4*” implements both “*DetectServiceDenial*” and “*LimitAccess*” tactics.

6 THREATS TO VALIDITY

EVSec and the two case studies are subject to limitations and threats to validity, that we categorize here according to three types of threats: internal, external, and construct.

In terms of *internal validity*, there is a risk related to how we describe the security scenario trigger. In our approach, the security exception path is triggered by the last responsibility before the or-join. This choice is made to preserve the dynamic semantics of the produced UCM. However, the trigger might be another responsibility occuring before the or-Join. As a result, additional semantic rules are required to obtain correct correlations and to filter out superfluous events. Another possible risk may be related to the filtering and tagging of the log entries. Indeed, in some cases it is difficult to figure out what to search for in a log (using keywords to distinguish between security and normal path operations), as there is no well-defined mapping between log messages and observed behavior.

In terms of *external validity*, the applicability of our approach was demonstrated using two real-world security features. The produced scenarios covers six security tactics only, namely, *IdentifyActors*, *AuthenticateActors*, *AuthorizeActors*, *LockComputer*, *DetectServiceDenial*, and *LimitAccess*. Additional case studies that target the remaining tactics are needed to generalize the applicability of *EVSec* approach. Furthermore, the used security features are part of routers running Cisco IOS software. The generalization to other suppliers’ security features has to be explored.

As for the *construct validity*, scalability may hinder the readability of the produced UCM. As the log size increases, the number

Component	Merged and filtered log	Execution Phase	Execution Label
Target	*Jun 10 08:43:40.015: %PARSER-5-CFGLOG_LOGGEDCMD: User:tty1 logged command:interface POS3/0 *Jun 10 08:43:41.939: %PARSER-5-CFGLOG_LOGGEDCMD: User:tty1 logged command:shutdown *Jun 10 08:43:40.015: %PARSER-5-CFGLOG_LOGGEDCMD: User:tty1 logged command:interface POS3/0 *Jun 10 08:43:41.939: %PARSER-5-CFGLOG_LOGGEDCMD: User:tty1 logged command:shutdown	1	User
Target	*Jun 10 08:43:41.955: %OSPF-5-ADJCHG: Process 1, Nbr 3.3.3.3 on POS3/0 from FULL to DOWN, Neighbor Down: Interface down or detached *Jun 10 08:43:43.927: %LINK-5-CHANGED: Interface POS3/0, changed state to administratively down *Jun 10 08:43:44.927: %LINEPROTO-5-UPDOWN: Line protocol on Interface POS3/0, changed state to down	2	Network
Target	*Jun 10 08:44:57.935: %SEC-6-IPACCESSLOGDP: list 100 denied icmp 1.1.1.1 -> 4.4.4.4 (0/0), 20 packets	3	Sec
Target	*Jun 10 08:45:05.687: %PARSER-5-CFGLOG_LOGGEDCMD: User:tty1 logged command:no shutdown	4	User
Target	*Jun 10 08:45:07.675: %LINK-3-UPDOWN: Interface POS3/0, changed state to up *Jun 10 08:45:08.683: %LINEPROTO-5-UPDOWN: Line protocol on Interface POS3/0, changed state to up *Jun 10 08:45:13.135: %OSPF-5-ADJCHG: Process 1, Nbr 3.3.3.3 on POS3/0 from LOADING to FULL, Loading Done	5	Network

Figure 7: Log relative to the uRPF feature

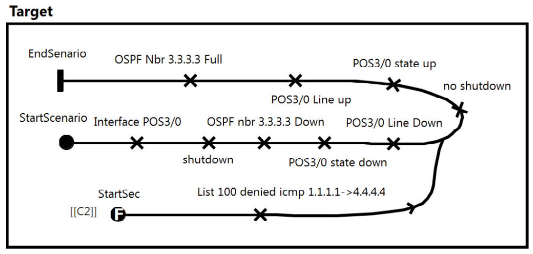


Figure 8: uRPF scenario log visualization

of phases increases. Although the stub/plugin mechanism offers an effective encapsulation technique, models may quickly get cluttered with overlapping paths. This problem can be alleviated by narrowing the timestamp log window.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced *EVSec* approach in order to extract and visualize security scenarios from logs collected from systems running security features. Logs are first merged, filtered (based on a user defined set of filtering keywords), tagged and segmented (based on a user-defined dictionary) into execution phases, then visualized as UCM security scenarios. The applicability of *EVSec* has been shown by applying it to two real-world security features, namely, Cisco IOS Login block and Cisco Unicast Reverse Path Forwarding (uRPF).

As future work, we plan to validate empirically (through expert surveys) our resulting UCM models with respect to their understandability and usefulness. In addition, we intend to examine the design of semantic rules in order to better correlate the various execution phases. This would allow for more precise UCM-based security models.

REFERENCES

[1] Thoms Ball. 1999. The Concept of Dynamic Analysis. *SIGSOFT Softw. Eng. Notes* 24, 6 (Oct. 1999), 216–234. https://doi.org/10.1145/318774.318944
[2] Len Bass, Paul Clements, and Rick Kazman. 2012. *Software Architecture in Practice* (3rd ed.). Addison-Wesley Professional.
[3] Cisco Systems. 2004. *Internetworking Technologies Handbook.* Cisco Press. http://books.google.com.sa/books?id=3Dn9KlIVM_EC
[4] Cisco Systems. 2011. Unicast Reverse Path Forwarding Concepts and Configuration. https://www.ciscopress.com/articles/article.asp?p=1725270 Last accessed, Feb 2022.
[5] GNS3 Technologies Inc. 2021. Graphical Network Simulator, GNS3. http://www.gns3.com/ Last accessed, December 2021.
[6] John Haggerty and Thomas Hughes-Roberts. 2014. Visualization of System Log Files for Post-incident Analysis and Response. In *Human Aspects of Information Security, Privacy, and Trust (LNCS, Vol. 8533)*, Theo Tryfonas and Ioannis G. Askoxylakis (Eds.). Springer, 23–32. https://doi.org/10.1007/978-3-319-07620-1_3
[7] Jameleddine Hassine. 2015. Describing and assessing availability requirements in the early stages of system development. *Softw. Syst. Model.* 14, 4 (2015), 1455–1479. https://doi.org/10.1007/s10270-013-0382-0
[8] Jameleddine Hassine and Abdelwahab Hamou-Lhadj. 2014. Toward a UCM-Based Approach for Recovering System Availability Requirements from Execution Traces. In *System Analysis and Modeling: Models and Reusability - 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8769)*. Springer, 48–63. https://doi.org/10.1007/978-3-319-11743-0_4
[9] Jameleddine Hassine and Abdelwahab Hamou-Lhadj. 2015. Describing Early Security Requirements Using Use Case Maps. In *17th International SDL Forum, Berlin, Germany, October 12-14, 2015, Proceedings*. Springer, 202–217. https://doi.org/10.1007/978-3-319-24912-4_15
[10] Jameleddine Hassine, Abdelwahab Hamou-Lhadj, and Luay Alawneh. 2018. A framework for the recovery and visualization of system availability scenarios from execution traces. *Inf. Softw. Technol.* 96 (2018), 78–93. https://doi.org/10.1016/j.infsof.2017.11.007
[11] IBM Security. 2020. Cost of a Data Breach Report 2020. https://www.ibm.com/security/digital-assets/cost-data-breach-report/1Cost%20of%20a%20Data%20Breach%20Report%202020.pdf Last accessed, Dec 2021.
[12] ITU-T. 2018. Recommendation Z.151 (10/18), User Requirements Notation (URN) Language Definition, Geneva, Switzerland. http://www.itu.int/rec/T-REC-Z.151/en
[13] jUCMNav v7.0.0. 2016. jUCMNav Project (tool, documentation, and meta-model). http://softwareengineering.ca/jucmnav
[14] Steven P. Reiss. 2006. Visualizing Program Execution Using User Abstractions. In *Proceedings of the 2006 ACM Symposium on Software Visualization* (Brighton, United Kingdom) *(SoftVis '06)*. ACM, New York, NY, USA, 125–134. https://doi.org/10.1145/1148493.1148512
[15] Jan Svacina, Jackson Raffety, Connor Woodahl, Brooklynn Stone, Tomás Cerný, Miroslav Bures, Dongwan Shin, Karel Frajták, and Pavel Tisnovsky. 2020. On Vulnerability and Security Log analysis: A Systematic Literature Review on Recent Trends. In *RACS '20: International Conference on Research in Adaptive and Convergent Systems, Gwangju, Korea, October 13-16, 2020*, Tomás Cerný and Juw Won Park (Eds.). ACM, 175–180. https://doi.org/10.1145/3400286.3418261
[16] Sheldon Teelink and Robert F. Erbacher. 2006. Improving the Computer Forensic Analysis Process Through Visualization. *Commun. ACM* 49, 2 (Feb. 2006), 71–75. https://doi.org/10.1145/1113034.1113073
[17] Andy Zaidman. 2006. Scalability Solutions for Program Comprehension Through Dynamic Analysis. In *Proceedings of the Conference on Software Maintenance and Reengineering (CSMR '06)*. IEEE Computer Society, Washington, DC, USA, 327–330. http://dl.acm.org/citation.cfm?id=1116163.1116422
[18] Yanping Zhang, Yang Xiao, Min Chen, Jingyuan Zhang, and Hongmei Deng. 2012. A survey of security visualization for computer network logs. *Secur. Commun. Networks* 5, 4 (2012), 404–421. https://doi.org/10.1002/sec.324