# Text Mining for Modeling Cyberattacks

Steven Noel[1,2]
*The MITRE Corporation, McLean, VA, United States*
[1]*Corresponding author: e-mail: snoel@mitre.org*

**ABSTRACT**
This chapter examines how natural language processing can be applied for building rich models for cybersecurity analytics. For this, it applies text mining to the natural-language content of Common Attack Pattern Enumeration and Classification (CAPEC™), a standardized corpus of cyberattack patterns. We adopt a vector-space model in which CAPEC attack patterns are treated as documents with term vectors. This provides a space in which to define distance measures, such as for retrieving attack patterns through term queries or finding clusters of related attack patterns. Analysis of clustering patterns, i.e., cluster hierarchies (clusters within clusters) is aided through tree visualization techniques. These analytic and visual techniques provide a range of capabilities for leveraging the content and relationships in CAPEC, e.g., for building more complex security models such as network attack graphs.

**Keywords:** Text mining, Cybersecurity, Hierarchical clustering, Attack graphs

## 1 INTRODUCTION

Maintaining resilience in cyberspace requires considering the full range of potential attacks against weaknesses in defended systems. Attackers often have to exploit only a single exposed vulnerability to gain a foothold on a network, from which other vulnerabilities are exposed. Security hygiene and attack resilience mechanisms are necessary but not sufficient. One needs comprehensive attack models to understand the scope and limitations

2. Approved for Public Release; Distribution Unlimited. Case Number 17-4570. The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions, or viewpoints expressed by the author.

**Handbook of Statistics.** https://doi.org/10.1016/bs.host.2018.06.001

of security capabilities, and to provide some assurance about the completeness of the defense posture.

Such attack models need to take the attacker's perspective, to capture the approaches for exploiting systems. Still, deep understanding of the variety of potential attacks is time-consuming and requires specialized skills. If common cyberattack patterns can be captured in a formalized and standard way, this valuable knowledge can be shared with the cybersecurity community.

Common Attack Pattern Enumeration and Classification (CAPEC™) (The MITRE Corporation, 2018a) is a publicly available, community-developed catalog of common attack patterns. CAPEC attack patterns capture knowledge about classes of attacks against various cyberspace vulnerabilities. Each attack pattern includes details about specific phases of the attack, the vulnerable attack surface, the resources required by the attacker, and ways to mitigate the attack. CAPEC thus provides a formal mechanism for identifying, collecting, refining, and sharing attack patterns.

CAPEC attack patterns are organized in a hierarchy from general to specific, providing various levels of abstraction to match analytic requirements. The hierarchy is specified through parent–child relationships among attack patterns, where a child is a specialization of its parent. Working with CAPEC requires understanding not only the hierarchical relationships of attack patterns at large but also the structure and content of individual attack patterns. Each CAPEC attack pattern includes a variety of elements of various data modalities, including categorical data, cross-references to other standards, and natural language.

In this chapter, we examine a number of approaches for leveraging CAPEC in building cyberattack models. A particular focus is mining CAPEC's natural-language content. For this, we define term vectors for each attack pattern, forming a vector space for computing distances (inverse similarities) between pairs of attack patterns. We apply these distances for a kind of significance ranking of attack pattern matches to query term vectors. We also apply these distances for hierarchical clustering, to discover nested groups of related attack patterns. These techniques lay a foundation for potential future work in attack pattern classification, e.g., as an alternative to manual taxonomy building or for chaining individual attack patterns into more complex multistep attacks.

Section 2 gives an overview of the various types of information provided for each CAPEC attack pattern. Section 3 then applies attack patterns to some security scenarios, showing how information of various types (categorical data, taxonomic structure, and natural language) can each play analytic roles. Section 4 applies text mining techniques to CAPEC natural-language content, i.e., defining a term-vector space for computing attack pattern distances, and applying those distances for query result ranking and attack pattern clustering. Section 5 gives examples that span multiple attack patterns, motivating the need for interpattern correlation based on natural language. Section 6 proposes various interactive visualization methods for exploring hierarchical relationships among attack patterns. Section 7 then describes software architecture for implementing this suite of analysis and visualization capabilities.

## 2 ANATOMY OF AN ATTACK PATTERN

Effective prevention and resilience techniques require understanding the range of potential adversary attack capabilities. For cybersecurity research, it is crucial that we define the assumed attack model, to convey the scope and limitations of our approaches. There is a wealth of information provided in CAPEC attack patterns that can be leveraged for building cyberattack models, especially content expressed in natural language.

CAPEC is part of MITRE's Making Security Measurable (Martin, 2009; The MITRE Corporation, 2013), a collection of collaborative initiatives for shared information, languages, and processes for information security. These initiatives encompass software assurance, threat analysis, vulnerability management, malware protection, intrusion detection, incident coordination, and other areas of security. Collaborators include teams from U.S. Department of Homeland Security (DHS), Defense Information Systems Agency (DISA), National Institute of Standards and Technology (NIST), Internet Engineering Task Force (IETF), and many others from government, academia, and industry (The MITRE Corporation, 2014).

In CAPEC, standardized attack patterns describe methods for exploiting system weaknesses. This gives us the attacker's perspective, and guidance on ways to mitigate the attack effects. Attack patterns are reminiscent of the concept of design patterns, but applied for cyberattack (Hoglund and McGraw, 2004; Moore et al., 2001). CAPEC attack patterns are built from in-depth analysis of real-world exploit examples. CAPEC captures this knowledge in a standard way, bringing considerable value for software security considerations through all phases of research and development. With CAPEC, attack patterns become actionable, shared knowledge. The detailed analysis for abstracting attack patterns requires specialized skills and deep experience. In practice, they are created by a relatively small number of specialists, e.g., within the security operations and incident response communities. Attackers are creative, actively collaborate, and have powerful tools (Chickowski, 2017). CAPEC helps close the knowledge gap between attacker capabilities and security solutions.

A CAPEC attack pattern is an abstraction for understanding how particular classes of attacks against vulnerable computer systems are executed. Each pattern defines the challenges that an attacker faces, and techniques for meeting those challenges. Attack patterns also give methods for mitigating the attack, indicators that warn of the attack, and references to related entities such as reported vulnerabilities and specific categories of software weaknesses.

Table 1 shows the core information for attack patterns in CAPEC (Barnum and Sethi, 2007). This is the basic information generally provided for all attack patterns. CAPEC attack patterns often include additional information such as in Table 2.

CAPEC attack patterns must strike a balance in providing the appropriate level of abstraction. The abstractions cannot be too generic or theoretical. Instead, they must identify the specific weakness that is being attacked and

**TABLE 1** Core Information for CAPEC Attack Patterns

| Field | Modality | Description |
|---|---|---|
| Methods | Categorical | Mechanism employed for carrying out the attack, e.g., malicious data entry, maliciously crafted file, protocol corruption |
| Prerequisites | Natural language | Conditions, functionality, characteristics, etc. that the target must have for the attack to succeed |
| Skills | Natural language | Level of skill or specific knowledge that the attacker must have to execute the attack, on a rough scale as well as in contextual detail |
| Resources | Natural language | The particular resources (e.g., CPU cycles, IP addresses, tools, time) that are required to carry out the attack |
| Context | Categorical | The technical contexts (e.g., platform, operating system, language, architectural paradigm) to which the attack is relevant |
| Consequences | Categorical | Results of the attack, in terms of scope (confidentiality, integrity, or availability) and specific technical impact |
| Solutions | Natural language | The recommended approaches to mitigate the attack, through remediation and/or resiliency |
| Vulnerabilities | Identifiers | Specific vulnerabilities (referenced via standard identifiers) for particular systems that may be exploited by the attack |
| Weaknesses | Identifiers | Underlying issues that may cause vulnerabilities (referenced via standard identifiers) |

how malicious input is provided to the target. On the other hand, CAPEC attack patterns are not so specific that they only apply to a particular application. Too much specificity can be dangerous to disclose and will likely provide limited benefit to defenders.

Fig. 1 shows the structure of the key elements of a CAPEC attack pattern. This structure is hierarchical (a tree), reflecting how the data elements for CAPEC are organized in eXtensible Markup Language (XML). In Fig. 1, nodes of the tree followed by square brackets indicate multiple instances of the element. Elements are styled according to the mode of data—green ovals for standard identifiers, multicolored bars for fixed categories, and red script glyphs for natural-language text.

Included in Fig. 1 are the challenges that must be met for the attack to be successful. This includes the steps for each phase of the attack, e.g., exploration to understand the target environment, experimentation for matching the attack

**TABLE 2** Additional Information for CAPEC Attack Patterns

| Field | Modality | Description |
|---|---|---|
| Examples | Natural language | Explanatory or demonstrative examples of the type of attack, to provide more practical and concrete context |
| Warnings | Natural language | Conditions indicating that the attack is imminent, in progress, or has occurred |
| Probing | Natural language | Techniques to probe and reconnoiter a potential target for the attack |
| Vector | Natural language | Mechanism and format for attack input, including grammar, syntax, field positions, data ranges, etc. |
| Payload | Natural language | Code, configuration, or other data executed or activated as part of the attack |
| Zone | Natural language | Area within the target capable of executing or activating the payload, e.g., command interpreter, machine code in a buffer, client browser, system call, etc. |
| Likelihood | Categorical | Typical degree of likelihood of the attack being successful (Likert scale) |
| Severity | Categorical | Typical level of severity resulting from the attack (Likert scale) |
| Phases | Natural language | Steps required for each phase of the attack, including exploration of the target environment, experimentation for matching the attack to the target, and exploitation of the target's weakness |

to the chosen target, and exploitation of the target weakness. It also includes technical prerequisites that must be met, attacker resources required, and techniques for probing the environment.

The technical context in Fig. 1 is the set of system architectures, application frameworks, operating system platforms, and/or programming languages applicable to the attack. Mechanisms of attack include the injection vector, its payload, and the zone (attack surface) in which the vector is activated. The context and methods of attack are given as general categories, vs specific description in natural-language text. This supports selection of attacks via an enumeration of known categories with clear semantics.

As shown in Fig. 1, a CAPEC attack pattern also includes knowledge about the impact of the attack. This content is given as categorical values. The likelihood of the attack succeeding and the attack's severity are categorized as very low, low, medium, high, or very high. A set of attack consequences is given, with each consequence having a particular scope (integrity, access control,

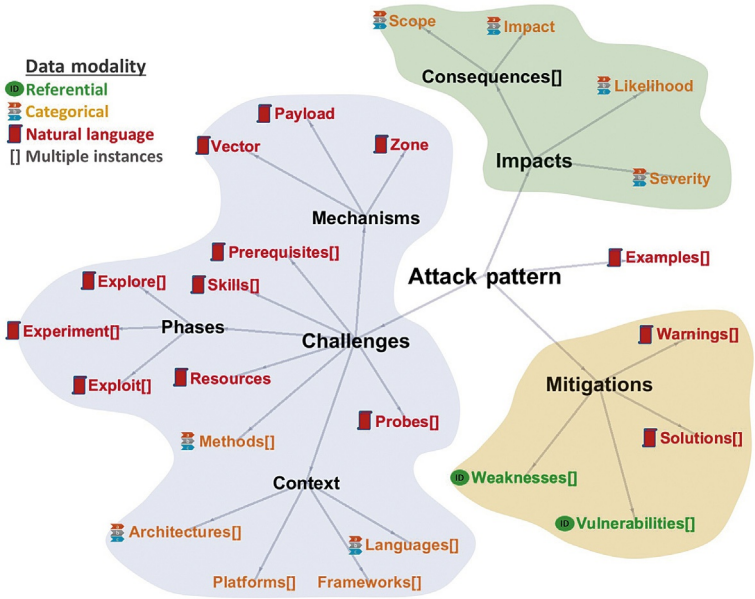


**FIG. 1** Structure of a CAPEC attack pattern.

authentication, availability, etc.) and technical impact (gain privileges, read files, modify memory, etc.). Example instances of the attack pattern are also given. Various parts of a CAPEC attack pattern support efforts for mitigating the attack. This includes solutions for removing the underlying weakness, or at least mitigating the effects of the attack, as well as attack warning indicators.

CAPEC attack patterns also include references to related security standards such as Common Weakness Enumeration (CWE™) (The MITRE Corporation, 2018c) and Common Vulnerabilities and Exposures (CVE®) (The MITRE Corporation, 2018b). CWE references describe the actual weakness that enable an attack pattern, so that the root causes can be addressed, e.g., through software assurance. CVE references identify system vulnerabilities that have been discovered that are susceptible to the attack.

## 3 APPLYING ATTACK PATTERNS TO SCENARIOS

CAPEC is an enumeration and taxonomic classification of cyberattack patterns, expressed in a standardized way. Because of their rich information content, expressed through a mixture of categorical data, referential data, and natural language, CAPEC attack patterns support a wide range of analytic use cases. This section considers some example applications of CAPEC for cyberattack modeling. In particular, we analyze the CAPEC corpus to find attack patterns of relevance to particular attack scenarios.

### 3.1 Resource Consumption Attacks

An important role for CAPEC is providing a comprehensive enumeration of common attacks, based on some criteria of interest. As an illustrative example, consider attack patterns involving resource consumption attacks, which fall under the category of denial-of-service (DoS) attacks (US-CERT, 2013).

As an analytic criterion, resource consumption appears in CAPEC attack patterns as categories of technical impact. In particular, the three categories for *DoS: resource consumption* are CPU, memory, and other. Table 3 shows all the CAPEC attack patterns for each of these categories.

In Table 3, there are two CAPEC attack patterns that have multiple categories of technical impact. In particular, attack patterns *XML Entity Expansion* (CAPEC-197, highlighted in blue) and *Leverage Alternate Encoding* (CAPEC-267, highlighted in pink) are each attacks against three kinds of system resources (CPU, memory, and other). The other attack patterns are consumption attacks against only a single kind of system resource. This analysis of categorical data from CAPEC not only finds the members of a certain class of attacks (those resulting in excessive resource consumption) but also identifies attacks that impact multiple resource types.

Another important property of attack patterns is their place in the CAPEC taxonomic "family tree." One analytic strategy it to examine the taxonomic relationships to find more general classes of attacks, starting from specific attacks of interest. For example, the attack pattern *XML Ping of the Death* (CAPEC-147) in Table 3 is part of the attack pattern taxonomic hierarchy in Fig. 2.

The root of this hierarchy is general flooding attacks, in which an attacker consumes a victim's resources through large numbers of rapid interactions, exploiting weakness in rate limiting or flow control. These kinds of attacks can occur at various layers of the internet protocol suite, e.g., internet layer (ICMP), transport layer (TCP, UDP), and application layer (SSL, HTTP).

As CAPEC describes, such flooding attacks may exploit implicit weaknesses in the protocol itself. For example, in the TCP protocol, servers need to maintain state information about their connections, so that large numbers of connection requests exhaust server resources. Similarly, large numbers of session-based HTTP GET requests consume web server resources. Since these are legitimate requests, this kind of attack can be difficult to detect. In the case of SSL flooding, because of the asymmetric processing requirements for servers vs clients in creating a secure connection, an attacker with a modest client machine can tie up disproportionately large server resources.

CAPEC also points out that while the UDP protocol is stateless, firewalls often save state information for each connection to a UDP service—thus UDP flood attacks can exhaust firewall resources. These kinds of attacks can also target UDP-based services such as DNS or VoIP. The lack of sessions in UDP makes it easier to spoof the source of attacks.

**TABLE 3** Resource Consumption (DoS) Attacks in CAPEC

| Resource | CAPEC ID | Attack Pattern Name |
|---|---|---|
| CPU | 29 | Leveraging Time-of-Check and Time-of-Use Race |
| | 197 | XML Entity Expansion |
| | 267 | Leverage Alternate Encoding |
| Memory | 14 | Client-side Injection-induced Buffer Overflow |
| | 69 | Target Programs with Elevated Privileges |
| | 72 | URL Encoding |
| | 78 | Using Escaped Slashes in Alternate Encoding |
| | 82 | Violating Implicit Assumptions Regarding XML Content |
| | 99 | XML Parser Attack |
| | 197 | XML Entity Expansion |
| | 230 | XML Nested Payloads |
| | 231 | XML Oversized Payloads |
| | 267 | Leverage Alternate Encoding |
| | 484 | XML Client-Side Attack |
| Other | 2 | Inducing Account Lockout |
| | 5 | Analog In-band Switching Signals (aka Blue Boxing) |
| | 25 | Forced Deadlock |
| | 27 | Leveraging Race Conditions via Symbolic Links |
| | 64 | Using Slashes and URL Encoding Combined |
| | 147 | XML Ping of the Death |
| | 197 | XML Entity Expansion |
| | 267 | Leverage Alternate Encoding |

## 3.2 Attacks for Introduction-Based Routing

The previous section examines CAPEC attack patterns based on *attacker-oriented* criteria. This section analyzes attack patterns of relevance to a particular *system to defend*. In particular, it considers attack patterns of relevance to Introduction-Based Routing (IBR) (Frazier et al., 2011). The goal is to discover attack patterns that are potentially effective against IBR, along with potential mitigations.

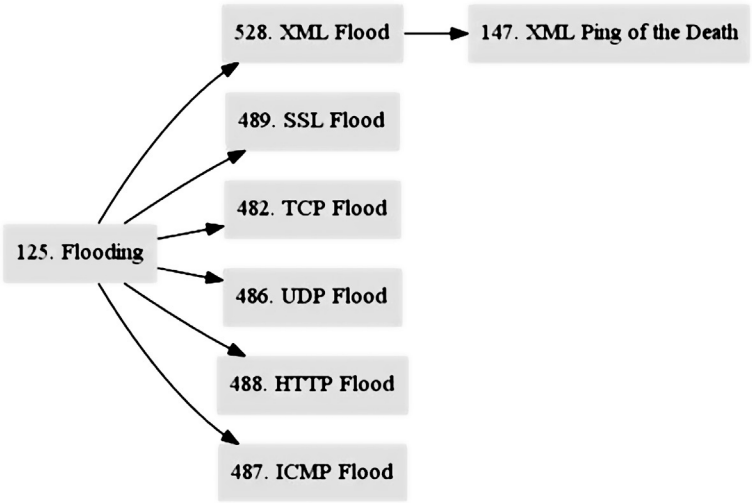


**FIG. 2** CAPEC hierarchy for flooding attacks.

IBR is a framework for secure connectivity management, in which nodes maintain the reputations of other nodes that connect with them. It provides a protocol for sharing feedback about node behavior and requires that network communication paths be established through introductions from participating IBR nodes. These introductions are granted based on node reputation, with is derived from shared feedback. IBR nodes maintain their own policies for maintaining reputations, giving feedback, and providing introductions.

Simulator software is available to study IBR behavior under a range of conditions. It includes a model for attack and defense, in which IBR nodes send either benign or malicious messages to other nodes. The attacker nodes act independently, and choose their victims randomly. We focus on the assumptions implied by the IBR simulator to guide the selection of CAPEC attack patterns.

In particular, in the IBR simulator, messages are sent between client nodes and server nodes. Clients initiate secure (encrypted) connections with servers. Once a connection is established between a client and server, they exchange a number of messages. A particular set of nodes (clients and/or servers) are designated as attackers. An attacker sends either good messages or attack messages, according to a given attack rate.

Thus, for attack patterns relevant to IBR, we seek attacks against client–server architecture. We want to limit our selection to attacks that actually compromise the client or server victim, e.g., allowing the attacker to execute arbitrary code.

In our process for selecting attack patterns, we also wish to exclude attacks that involve a third middle man machine (spoofing, cross-site scripting, etc.) other than the client and server, under the assumption that IBR allows only communications between the client and server, via secure connection. We

also seek to exclude attacks that involve repetitive attempts (e.g., brute force), under the assumption that they will quickly be detected and disconnected from the IBR network.

In CAPEC (version 2.5), there are 453 total attack patterns. Of those, 17 lack descriptions. We exclude them, leaving 436 attack patterns to consider. As described in the previous section, attack patterns may include knowledge of attack phases, e.g., exploration, experimentation, and exploitation. We select only those patterns that have an exploitation phase defined. This reduces the 436 patterns to 103 patterns.

Refining this further, attack patterns have a technical context of a particular set of architecture paradigms (client–server, SOA, web, etc.). Consistent with the IBR simulator attack/defense model, we select only those patterns that have architecture paradigms all, client–server, or n-tier. This reduces the 103 patterns to 95. In CAPEC, attack patterns are assigned one of three levels of abstraction: meta, standard, or detailed. Meta is the highest level (broader), and detailed is the lowest level (more focused). We exclude the meta-level patterns, which reduces the selection from 95 to 94 (64 standard and 30 detailed).

Attack patterns have defined purposes, i.e., exploitation, obfuscation, penetration, or reconnaissance. We select only those having purpose of exploitation or penetration. This reduces the standard patterns from 64 to 57. All 30 of the detailed patterns have those purposes.

Attack patterns may define their technical impact. Here are the possible choices:

1. Alter execution logic
2. Bypass protection mechanism
3. DoS
   - **(a)** Amplification
   - **(b)** Crash/exit/restart
   - **(c)** Instability
   - **(d)** Resource consumption (CPU)
   - **(e)** Resource consumption (memory)
   - **(f)** Resource consumption (other)
4. Execute unauthorized code or commands
5. Gain privileges/assume identity
6. Hide activities
7. Modify
   - **(a)** Application data
   - **(b)** Files or directories
   - **(c)** Memory
8. Read
   - **(a)** Application data
   - **(b)** Files or directories
   - **(c)** Memory
9. Unexpected State
10. Varies by context

We select only those patterns with technical impact of "gain privileges/assume identity," "execute unauthorized code or commands," "modify memory," or "unexpected state." This corresponds to the requirement of attacks that eventually lead to being able to execute code on the victim. Selecting by technical impact, this reduces the standard patterns from 57 to 54. Detailed patterns are reduced from 30 to 28.

Attack patterns define the likelihood of their exploitation (very high, high, medium, low, or very low). We select only those patterns with medium, high, or very high likelihood of exploitation. This reduces the standard patterns from 54 to 47. Detailed patterns are reduced from 28 to 25. Attack patterns define the severity of their exploitation (very high, high, medium, low, or very low). We select only those patterns with high or very high severity. This reduces the standard patterns from 47 to 41. Detailed patterns remain at 25.

Through manual examination of each of the remaining (41 standard and 25 detailed) attack patterns, we create a "black list" of patterns to exclude. The criteria fall into the categories of "man in the middle" and "too loud" as described above. Excluding by black list reduces the number of standard attack patterns from 54 to 22 (Table 4), and detailed attack patterns from 28 to 18 (Table 5).

As an example of an attack within the assumptions of IBR, consider CAPEC-14 *Client-side Injection-induced Buffer Overflow*. This attack "exploits a buffer overflow vulnerability in targeted client software through injection of malicious content from a custom-built hostile service." Fig. 3 shows the steps involved in this kind of attack. In the IBR simulator, attackers can be either clients or servers. The CAPEC-14 attack pattern involves a malicious server that attacks clients connecting to it.

In CAPEC-14, the attack payload is "attacker-supplied data potentially containing malicious code" delivered to a client from an attacking server. For the impact of payload activation, CAPEC says "the most common are remote code execution or denial of service." Fig. 4 shows the consequences of this attack.

Attacker skill needed for denial of service is relatively low. CAPEC reports that "an attacker can simply overflow a buffer by inserting a long string into an attacker-modifiable injection vector." However, for remote code execution, "exploiting a buffer overflow to inject malicious code into the stack of a software system or even the heap requires a more in-depth knowledge and higher skill level."

This requires in-depth understanding of the memory architecture of the target platform, so that "when the function returns control to the main program, it jumps to the return address portion of the stack frame overwritten by the overflowed buffer [containing] a call to a privileged command or to malicious code." CAPEC reports that "there are potentially thousands of different ways data can propagate into a given system, thus these kinds of attacks will continue to be found in the wild."

Fig. 5 shows the probing techniques reported for this attack. This suggests some possible strategies for mitigating attacks through IBR. For example, we

**TABLE 4 Selected Standard-Level CAPEC Attack Patterns**

| ID | Name | Description |
|---|---|---|
| 100 | Overflow Buffers | Write past the boundaries of allocated buffer memory regions |
| 105 | HTTP Request Splitting | Insert additional HTTP requests |
| 13 | Subverting Environment Variable Values | Modify environment variables of target software |
| 135 | Format String Injection | Include formatting characters in a string input field on a target application |
| 15 | Command Delimiters | Concatenate commands onto a legitimate command |
| 159 | Redirect Access to Libraries | Exploit execution flow to an external library |
| 193 | PHP Remote File Inclusion | Execute code remotely through PHP improperly sanitizing call |
| 207 | Removing Important Functionality from the Client | Disable functionality on the client for secure operation |
| 21 | Exploitation of Session Variables, Resource IDs and other Trusted Credentials | Take advantage of software accepting input without verifying its authenticity |
| 26 | Leveraging Race Conditions | Exploit race condition in which multiple processes access resource concurrently |
| 43 | Exploiting Multiple Input Interpretation Layers | Supply input data with special characters that bypass input validation logic |
| 56 | Removing/short-circuiting "guard logic" | Avoid guard logic to access protected functionality or data |
| 6 | Argument Injection | Change behavior or state of application by injecting data or command syntax |
| 61 | Session Fixation | Establish session using session identifier provided by attacker |
| 63 | Simple Script Injection | Embed malicious script in content that will be served to web browsers |
| 69 | Target Programs with Elevated Privileges | Leverage bug to get arbitrary code to execute with elevated privileges |
| 72 | URL Encoding | Exploit ways of encoding URLs to abuse their interpretation |

**TABLE 4 Selected Standard-Level CAPEC Attack Patterns—Cont'd**

| ID | Name | Description |
|---|---|---|
| 74 | Manipulating User State | Modify state information in user-accessible locations to modify application flow |
| 76 | Manipulating Input to File System Calls | Manipulate inputs passed to file system calls in the OS |
| 77 | Manipulating User-Controlled Variables | Override environment variables without data sanitization |
| 88 | OS Command Injection | Leverage command injection to compromise underlying operating system |
| 92 | Forced Integer Overflow | Force integer variable to go out of range for executing arbitrary code |

**TABLE 5 Selected Detailed-Level CAPEC Attack Patterns**

| ID | Name | Description |
|---|---|---|
| 10 | Buffer Overflow via Environment Variables | Manipulate environment variables to overflow associated buffers |
| 14 | Client-side Injection-induced Buffer Overflow | Build hostile service that injects malicious content into client |
| 24 | Filter Failure through Buffer Overflow | Feed overly long input to software |
| 27 | Leveraging Race Conditions via Symbolic Links | Create symbolic link to target file not otherwise accessible |
| 34 | HTTP Response Splitting | Send HTTP request causing response to be interpreted by client as two separate responses |
| 41 | Using Meta-characters in E-mail Headers to Inject Malicious Payloads | Leverage meta-characters in email headers to inject improper behavior into email programs, e.g., containing scripts, enumerations, probes |
| 42 | MIME Conversion | Overflow buffer in the MIME conversion routine to gain control over mail server |
| 45 | Buffer Overflow via Symbolic Links | Create or manipulate symbolic link file such that its contents result in out of bounds data |

*Continued*

**TABLE 5 Selected Detailed-Level CAPEC Attack Patterns—Cont'd**

| ID | Name | Description |
|---|---|---|
| 46 | Overflow Variables and Tags | Craft malicious HTML page or configuration file that includes oversized strings |
| 47 | Buffer Overflow via Parameter Expansion | Provide input that is expanded in size during processing |
| 52 | Embedding NULL Bytes | Embed null bytes in input to stop input processing |
| 53 | Postfix, Null Terminate, and Backslash | When a terminal NULL is valid, using alternate representation to embed the NULL mid-string |
| 64 | Using Slashes and URL Encoding Combined to Bypass Validation Logic | Abuse multiple interpretation of URLs to bypass forbidden URL |
| 67 | String Format Overflow in syslog () | Inject malicious input in format string parameter of syslog function |
| 8 | Buffer Overflow in an API Call | Attack libraries or shared code modules, e.g., to embed malicious code in function call |
| 83 | XPath Injection | Craft input XPath expressions to bypass authentication in an XML database |
| 84 | XQuery Injection | Use XQuery to probe and attack server |
| 9 | Buffer Overflow in Local Command-Line Utilities | Exploit vulnerability in command-line utility to escalate privilege to root |

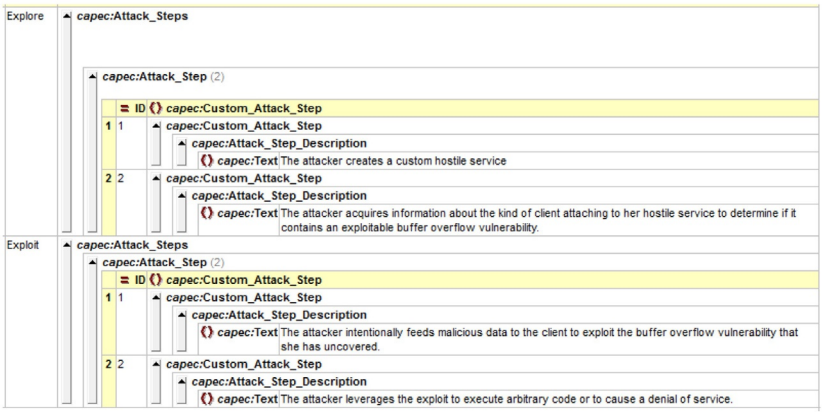


**FIG. 3** Attack steps for client-side buffer overflow injection (CAPEC-14).

| | capec:Consequence_Scope | capec:Consequence_Technical_Impact | capec:Consequence_Note |
|---|---|---|---|
| 1 | capec:Consequence_Scope (1)<br>Text<br>1 Confidentiality | Read memory | |
| 2 | capec:Consequence_Scope (1)<br>Text<br>1 Integrity | Modify memory | |
| 3 | capec:Consequence_Scope (1)<br>Text<br>1 Availability | DoS: resource consumption (memory) | capec:Consequence_Note<br>capec:Text Denial of Service |
| 4 | capec:Consequence_Scope (3)<br>Text<br>1 Confidentiality<br>2 Integrity<br>3 Availability | Execute unauthorized code or commands | capec:Consequence_Note<br>capec:Text Run Arbitrary Code |

**FIG. 4** Attack consequences for CAPEC-14.

| | capec:Description |
|---|---|
| 1 | capec:Text: The server may look like a valid server, but in reality it may be a hostile server aimed at fooling the client software. For instance the server can use honey pots and get the client to download malicious code. |
| 2 | capec:Description<br>capec:Text: Once engaged with the client, the hostile server may attempt to scan the client's host for open ports and potential vulnerabilities in the client software. |
| 3 | capec:Description<br>capec:Text: The hostile server may also attempt to install and run malicious code on the client software. That malicious code can be used to scan the client software for buffer overflow. |

**FIG. 5** Probing techniques for CAPEC-14.

| capec:Solution_or_Mitigation (8) | |
|---|---|
| | capec:Text |
| 1 | The client software should not install untrusted code from a nonauthenticated server. |
| 2 | The client software should have the latest patches and should be audited for vulnerabilities before being used to communicate with potentially hostile servers. |
| 3 | Perform input validation for length of buffer inputs. |
| 4 | Use a language or compiler that performs automatic bounds checking. |
| 5 | Use an abstraction library to abstract away risky APIs. Not a complete solution. |
| 6 | Compiler-based canary mechanisms such as StackGuard, ProPolice, and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution. |
| 7 | Ensure all buffer uses are consistently bounds-checked. |
| 8 | Use OS-level preventative functionality. Not a complete solution. |

**FIG. 6** Attack mitigations for CAPEC-14.

can detect the attacking server performing a port scan of the client, or attempting to install malicious scanning code on the client. The IBR protocol then lets the victim client report that misbehavior to participating IBR nodes. The reputation of the attacking server may then get sufficiently low that it is dropped from the IBR network. In this way, IBR facilitates the sharing of knowledge about bad network behavior, so that nodes benefit from the observations of others.

Fig. 6 shows the mitigations that CAPEC suggests for this attack. Some of these rely on good network hygiene, e.g., keeping patches current and not

installing untrusted code. Others rely on security tools such as vulnerability scanners and malware canaries, although these are not complete solutions. Other mitigations involve changes to the client software or operating system, which must be performed by the product developers. Overall, IBR can clearly provide an additional layer of protection against this kind of attack.

## 4 MINING ATTACK PATTERN TEXT

The analysis of CAPEC attack patterns in the previous section is based on attributes whose values are fixed categories. However, as described in Section 2, many of the other attributes of CAPEC attack patterns are given in natural-language text. While natural language provides freedom of expressiveness, it introduces challenges for analysis of attack patterns.

We treat each CAPEC attack pattern as a document containing text to be mined. We apply a vector-space model in which each document (attack pattern) is a vector whose components represent the terms (relevant words) in the document. From this representation, we define distances from documents to query terms, as well as interdocument distances. For information retrieval, we use query-document distances to rank documents by their strength of query match. We use document distances to cluster documents for exploratory analysis, i.e., discover groups of related attack patterns based on their language content. We do this through hierarchical clustering (clusters of clusters), showing clusters at ranges of distances.

### 4.1 Vector-Space Attack Pattern Model

A predominant approach for mining natural-language text is to model document collections as vector spaces (Wong et al., 1987). In such models, natural-language documents are represented as vectors, where each vector component (dimension) represents a term (word, phrase, etc.) that may appear in the document.

Thus, in applying a vector-space approach to the analysis of CAPEC attack patterns, we represent each attack pattern as a document. An attack pattern document has a vector of terms (words), in which a nonzero value for a vector component indicates the presence of that term in that document. The actual value of a component (its weight) indicates the strength of the term in the document (typically, computed from term frequency).

Vector and matrix operations can then be applied for analysis, e.g., to measure document similarity, cluster and classify documents, and retrieve documents matching queries. For building the document terms, we select three elements of CAPEC content: attack pattern name, summary, and prerequisites. These three particular attributes (written in natural-language text) express the general character of each attack, along with the conditions required for success.

Across the entire corpus of CAPEC patterns, for the selected attributes (name, summary, and prerequisites), there are 4313 unique words (terms).

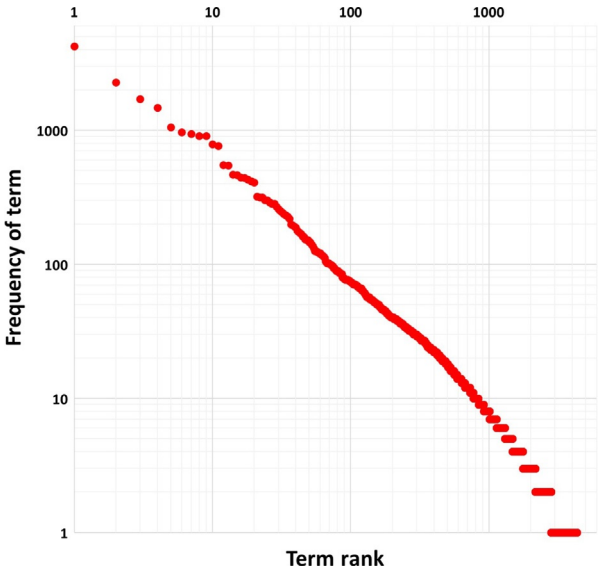


**FIG. 7** Term frequency distribution for CAPEC.

Fig. 7 shows the frequency of these words across the corpus. The words are ranked by frequency (ordinate axis), with the abscissa showing the number of times the word appears in the CAPEC attributes.

This distribution approximately follows the Zipf–Mandelbrot law (Mandelbrot, 1966; Zipf, 1949), in which term frequency is inversely proportional to term rank. This law has been related to general systems behavior in terms of fractal sequences of symbols, intermediate between deterministic predictability and complete chaos (Denisov, 1997). Empirically, this relation between word frequency and rank among all words is approximate and has been reformulated via a dual-regime distribution that more accurately accounts for rarer words (Montemurro, 2001).

On a double logarithmic scale as in Fig. 7, such an observed power-law distribution appears as a straight line with negative-unity slope. In a power-law distribution, a term of rank $k$ occurs $1/k$-th as often as the most-frequently-occurring word. This means that relatively few common words comprise much of the corpus, and there are many words that occur infrequently.

Fig. 8 shows the 30 most frequent words in our CAPEC text corpus (on a logarithmic scale). Intuitively, such common words provide no value in discerning the meaning of text content. Even words such as "attacker," "target," "attack," and "application" (which are relatively uncommon in more general text) provide little to distinguish meaning among CAPEC attack patterns, since they occur so often.

In selecting words to serve as terms in our document vectors, the usual approach is to exclude such frequently occurring noncontent-bearing words

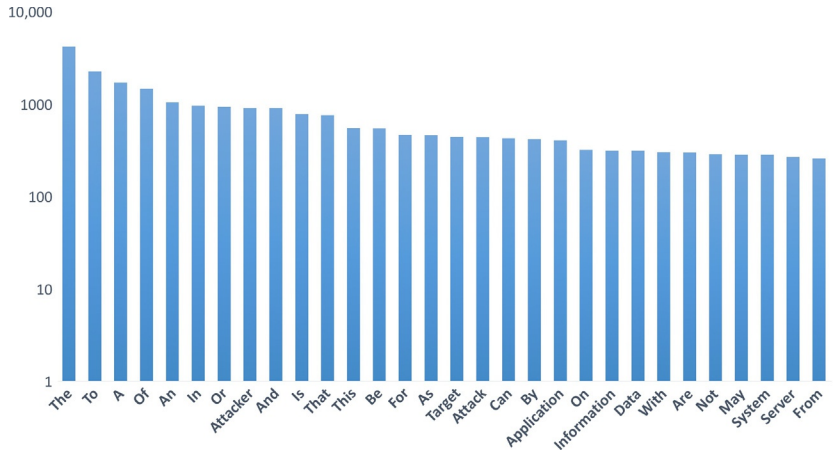


**FIG. 8** The 30 most frequent words in selected CAPEC text.

(so-called stop words) (Salton, 1983). In our experiments, we find that the set of stop words in Fig. 8 provides good results for our analysis.

In term-vector models, the terms for a document vector are given weights, denoting the relevant importance of each term within the document. Various weightings have been proposed, each having advantages and disadvantages in terms of efficiency and performance (e.g., for search results), often depending on characteristics of the text corpus (Lee et al., 1997; Salton and Buckley, 1996). These are often some variation of "term frequency-inverse document frequency (tf-idf )," in which a document term weight is proportional to its frequency within the document, and inversely proportional to its frequency over the entire corpus, so that more common words overall have reduced weight.

For a particular vector representation, we can then define distance between two document vectors. This allows us to compare document similarities, e.g., for clustering. We can also measure the distance of a query vector to our document vectors. Again, various document distance measures have been proposed. These are generally based on the extent of term overlap between two documents. The most common document similarity measure is cosine similarity. This measures similarity of two document vectors as the cosine of the angle between them (in the weighted-term space), through their dot (inner) product.

## 4.2 Query Relevance Distance

We can express a set of query words as a document term vector and measure the distance of the query vector to our CAPEC document vectors (attack patterns). In this case, the query vector represents concepts to be matched rather than another CAPEC attack pattern vector of terms. That is, the query vector does not represent term frequencies, just the occurrence of terms. Still,

intuitively we should consider higher frequencies of matched query words as stronger similarities (smaller distances).

We therefore define query-document distance as

$$d(\mathbf{q}, \mathbf{d}_j) = 1 - (\mathbf{q} \cdot \mathbf{d}_j)/(\mathbf{q} \cdot \mathbf{d}_{max}). \quad (1)$$

Here, $\mathbf{q} = (\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_t)$ is the query vector and $\mathbf{d}_j = (\mathbf{d}_{1,j}, \mathbf{d}_{2,j}, \ldots, \mathbf{d}_{t,j})$ is document vector $j$. The vector components for $\mathbf{q}$ and $\mathbf{d}_j$ are binary, i.e., $\mathbf{q}_i$, $\mathbf{d}_{i,j} \in \{0, 1\}$, representing the absence or presence of term $i$ in the query or document. The normalizing factor $\mathbf{q} \cdot \mathbf{d}_{max}$ is the largest inner product (best query match) over all documents. This yields the smallest distance $d(\mathbf{q}, \mathbf{d}_{max}) = 0$ for the document with the largest innerproduct similarity to the query, and the largest distance $d(\mathbf{q}, \mathbf{d}_j) = 1$ when document $d_j$ has no matching terms to query $q$.

As an example, consider the query vector $\mathbf{q}$ = (database, sql, query). Table 6 shows the resulting distances for CAPEC documents (attack patterns) $d_j$, for distances $d(q, d_j) < 0.6$. These are sorted in increasing order of distance (decreasing order of similarity) to the query. The rightmost column shows the number of query word matches for each attack pattern. The closest attack pattern to the query has 17 query word matches. The other query distances are the number of query word matches relative to the closest match.

**TABLE 6** CAPEC Matches to a Set of Query Terms

| ID | Name | Summary | Distance | Matches |
|---|---|---|---|---|
| 66 | SQL Injection | Exploit software that constructs SQL statements | 0 | database (6) sql (11) |
| 470 | Expanding Control over the OS from the Database | Leverage database access to compromise the operating system | 0.47 | database (7) sql(2) |
| 109 | Object Relational Mapping Injection | Leverage database access through Object Relational Mapping to inject SQL | 0.53 | database (4) sql(4) |
| 110 | SQL Injection through SOAP Parameter Tampering | Modify parameters of SOAP message for SQL injection | 0.59 | database (1) query (1) sql(5) |
| 136 | LDAP Injection | Craft LDAP query to inject SQL | 0.59 | query(6) sql(1) |

This illustrates a number of advantages for the vector-space representation of natural-language text. The vector-space model is straightforward, with its basis in well-established linear algebra. It allows one to express documents as vectors of weighted terms, beyond simply recording term occurrences. From this representation, we can compute a continuous range of query-document similarities/distances, e.g., for ranking documents according to strength of relevance. This naturally incorporates partial matching, vs supporting only strict Boolean matches.

## 4.3 Attack Pattern Distances

We can also apply our vector-space distances to measure distances (inverse similarities) among documents themselves, to cluster related documents. This allows us to discover groups of related CAPEC attack patterns based on their language content. To do this, we apply the standard tf-idf weighting (Robertson, 2004) to the document vectors. In this weighting, we have document vector $d_j = (d_{1,j}, d_{2,j}, \ldots, d_{t,j})$, where component $d_{i,j}$ represents term $i$ in document $j$. The weight $w_{j,i}$ for term $i$ is

$$w_{i,j} = f_{i,j} \cdot \log\left(\frac{|D|}{|\{d \in D | i \in D\}|}\right) \tag{2}$$

Here, $f_{i,j}$ is the frequency of term $i$ in document $j$, $|D|$ is the total number of documents in our text corpus, and $|\{d \in D | i \in D\}|$ is the number of documents that contain term $i$. The logarithmic base is arbitrary in the sense that it does not affect the ordering of document similarities. We use the base-10 logarithm, which is perhaps the most common choice.

This composite tf-idf weighting is composed of factors to enhance mutual recall and precision of terms when comparing documents. The term frequency $f_{i,j}$ enhances recall, in the sense that a document's frequently occurring terms should help in recalling it. On the other hand, terms that occur frequently in a large documents can lead to erroneous retrieval, so that precision suffers. The inverse document frequency $\log(|D|/|\{d \in D | i \in D\}|)$ addresses that by reducing weights for terms that occur frequently throughout all documents.

Given this weighted-term vector representation, we can compute the distance between a pair of documents. For this, we measure the angle (in high-dimensional term space) between the document vectors. Documents with a small angle between them are deemed more similar. We can leverage the identity $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta$, where $\theta$ is the angle between $\mathbf{a}$ and $\mathbf{b}$. We then define cosine similarity as

$$\text{similarity}_{\cos}(a,b) := \cos\theta = (\mathbf{a} \cdot \mathbf{b})/(|\mathbf{a}||\mathbf{b}|) \tag{3}$$

For parallel documents, $\cos 0 = 1$. For any other angle, $\cos\theta < 1$, down to the smallest value of $\cos(\pi/2) = 0$. Here, assume that $0 < \theta < \pi/2$, making the cosine positive, since the term weights are all positive (the angle between

them cannot be greater than 90 degrees). Thus, the cosine similarity ranges from zero (orthogonal document vectors) to unity (parallel document vectors).

Certain analyses such as clustering assume distances (dissimilarity), i.e., the complement of similarity. A proper distance metric (obeying the triangle inequality) requires the similarity to be expressed as an angle and normalized:

$$\text{similarity}_{\theta}(\mathbf{a},\mathbf{b}) := 1 - \frac{2\cos^{-1}(\text{similarity}_{\cos})}{\pi} \tag{4}$$

The document distance is then simply the complement:

$$\text{distance}(\mathbf{a},\mathbf{b}) = 1 - \text{similarity}_{\theta}(\mathbf{a},\mathbf{b}) = \frac{2\cos^{-1}(\text{similarity}_{\cos})}{\pi} \tag{5}$$

Fig. 9 shows the resulting distance matrix for the first 50 CAPEC attack patterns. Here, matrix element distance($\mathbf{d}_i$, $\mathbf{d}_j$) is the distance between documents (CAPEC attack patterns) $\mathbf{d}_i$ and $\mathbf{d}_j$. In the figure, we artificially set the self-distances (main diagonal) to the distance midpoint, i.e., $\text{distance}(\mathbf{d}_i,\mathbf{d}_i) = (d_{max} - d_{min})/2 + d_{min}$, where $d_{min}$ and $d_{max}$ are the minimum and maximum (respectively) distance values over all document pairs. This is simply to reduce the dynamic range of distances for plotting. For our analysis, the self-distances are the actual distance($\mathbf{d}_i$, $\mathbf{d}_i$) = 0. The distance scale denotes the 20-step graduation of distance values.

Fig. 10 shows the same distance matrix as in Fig. 9, this time as a surface plot. This shows more explicitly that most document distances are near unity (the greatest possible distance). In fact, the average document distance is 0.99. That is, most documents are dissimilar from others, and closer similarities are

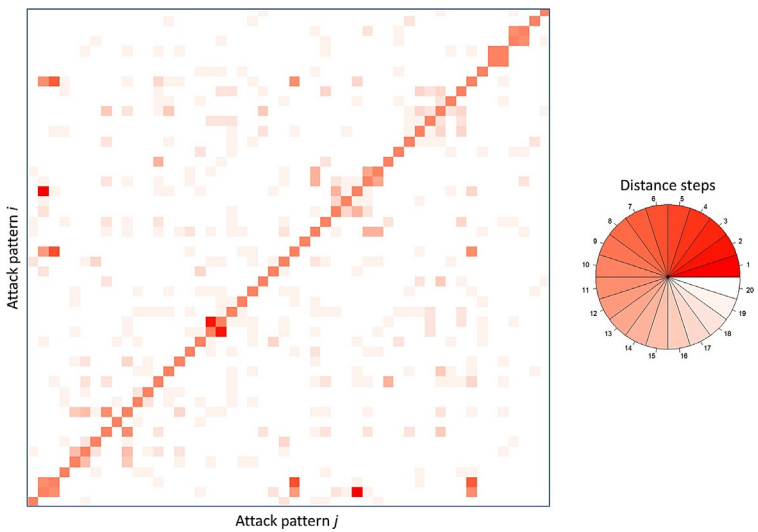


**FIG. 9** Document distances for first 50 CAPEC attack patterns.

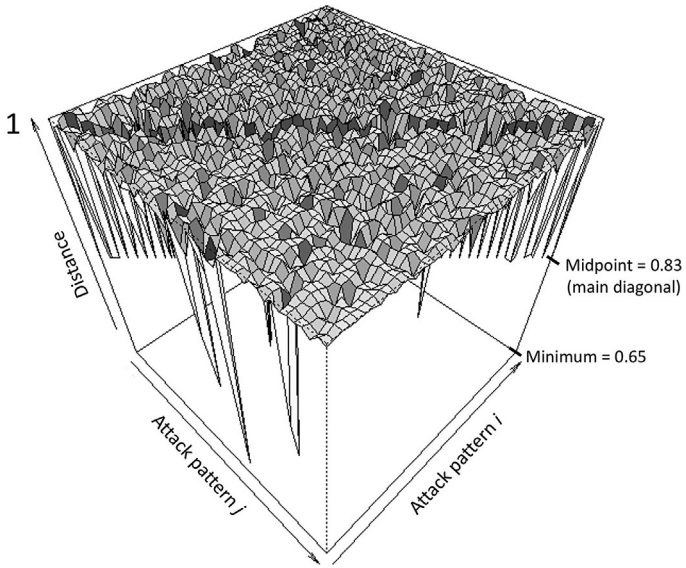


**FIG. 10** Document distances for first 50 CAPEC attack patterns (surface plot).

only for relatively few document pairs. This plot also more clearly shows the main diagonal values (artificially set to the distances midpoint). This also shows the minimum distance (closest document match) of 0.65.

## 4.4 Attack Pattern Clustering

Given distances between each pair of documents (CAPEC attack patterns), we can cluster them into groups of related attack patterns. Clustering is an important early step in exploratory data analysis. Many approaches to clustering have been proposed, although in a strict sense clustering is an ill-posed problem, in that there is no universal definition of what constitutes good clusters. The general strategy is to employ heuristics that are designed to give good results in general, followed by postanalysis of the results.

For exploratory analysis, it can also be helpful to apply *hierarchical* clustering methods. These methods build a hierarchy of clusters, i.e., compositional "clusters within clusters." We apply agglomerative clustering, in which pairs of clusters are merged as a merge distance threshold is relaxed. This allows us to see the trade off between numbers of clusters and cluster size (number of members), and the threshold distances at which the clusters form.

Various criteria are possible for defining distances between clusters based on the distances of the individual cluster members (documents). Three commonly applied criteria are *complete-linkage*, *average-linkage*, and *single-linkage* (Venables and Ripley, 1994), illustrated in Fig. 11.

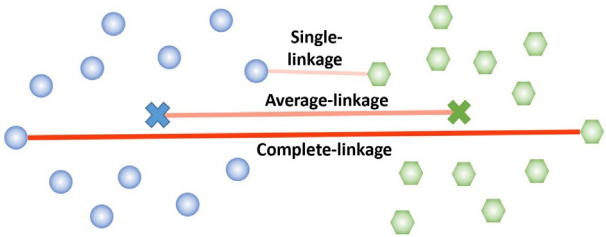


**FIG. 11** Distances between clusters.

As the clustering threshold increases, we merge clusters that have the closest distance between them (for a given threshold distance). This follows a given criterion for assigning overall cluster distances based on document distances:

- *Single-linkage*: The measure of distance between two clusters is the closest possible distance between documents in separate clusters.
- *Average-linkage*: Cluster distance is the average of distances between documents in separate clusters.
- *Complete-linkage*: Cluster distance is the furthest distance between objects in separate clusters.

Single-linkage is the weakest clustering criterion, i.e., only the two closest documents are sufficient to merge two clusters. Complete-linkage is the strongest criterion—all document distances across clusters must meet the threshold. Average-linkage has intermediate strength between those extremes. In single-linkage, documents merge with the closest member of a cluster, which ignores overall cluster structure. Complete-linkage favors compact clusters with smaller (and approximately equal) diameters. However, it is more sensitive to outliers, splitting clusters to accommodate documents that are distant from most others.

Once we compute our hierarchical clustering, we can visualize it as a *dendrogram*. This is a tree visualization, with the leaves of the tree representing the documents (CAPEC attack patterns) that we cluster. The top of the diagram shows the clustering threshold distance. At the smallest distances (starting at zero), no documents are sufficiently close to form clusters. At some threshold distance, some documents are close enough to combine into a cluster. For larger threshold distances, documents/clusters merge into combined clusters. The dendrogram shows the tree of cluster composition (clusters within clusters) at each level of distance threshold.

Fig. 12 shows the complete-linkage dendrogram for the first 50 CAPEC attack patterns. This is applied to the distances in Figs. 9 and 10.

In Fig. 12, each attack pattern is labeled by CAPEC identifier and name, i.e., C.*capecId.capecName*. The threshold distance is set such that eight clusters are formed. The clustering threshold (with the number of resulting

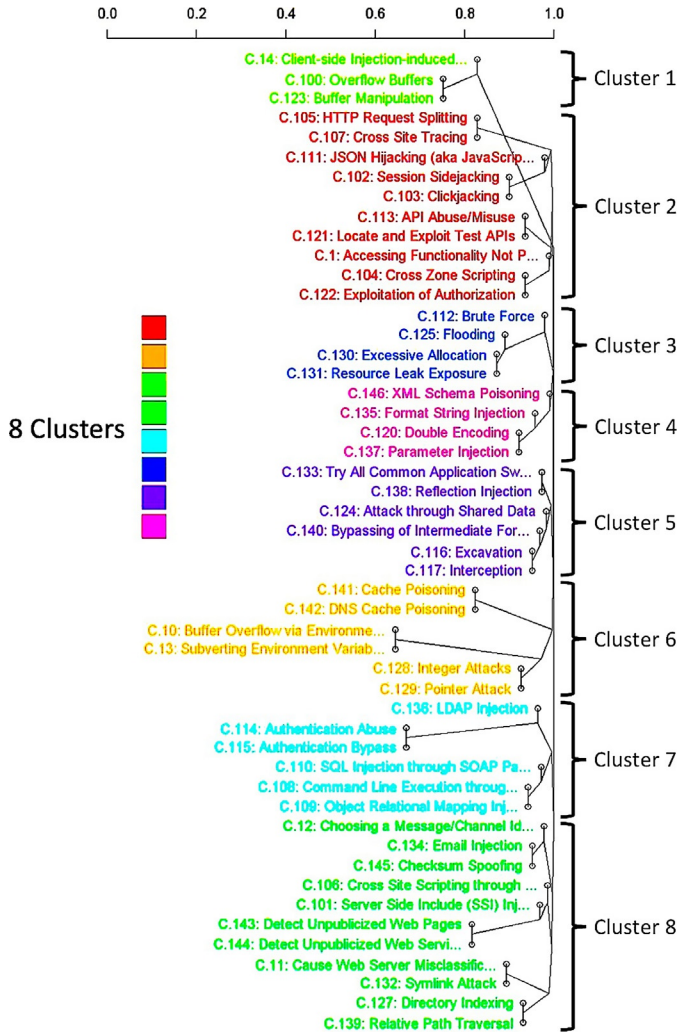


**FIG. 12** Cluster dendrogram for first 50 CAPEC attack patterns.

clusters) is a subjective choice based on our analysis goals and understanding of the problem domain.

An examination of the attack pattern clusters in Fig. 12 suggests that our analysis does a good job of identifying related attack patterns. For example, the two closest attack patterns in Fig. 12 (lowest cluster threshold, i.e., left-most merge) are CAPEC-10 *Buffer Overflow via Environment Variables* and CAPEC-13 *Subverting Environment Variable Values*. This pair of attack patterns has a document distance of 0.65. Table 7 shows the text content mined for these attack patterns. In examining this text, we can discern considerable use of common language.

**TABLE 7** Text Content for Two Most Similar Documents of Fig. 12

| ID | Name | Summary | Prerequisites |
|---|---|---|---|
| 10 | Buffer Overflow via Environment Variables | This attack pattern involves causing a buffer overflow through manipulation of environment variables. Once the attacker finds that they can modify an environment variable, they may try to overflow associated buffers. This attack leverages implicit trust often placed in environment variables. | This attack requires the following:<br>1. The application uses environment variables.<br>2. An environment variable exposed to the user is vulnerable to a buffer overflow.<br>3. The vulnerable environment variable uses untrusted data.<br>4. Tainted data used in the environment variables is not properly validated. For instance boundary checking is not done before copying the input data to a buffer. |
| 13 | Subverting Environment Variable Values | The attacker directly or indirectly modifies environment variables used by or controlling the target software. The attacker's goal is to cause the target software to deviate from its expected operation in a manner that benefits the attacker. | This attack requires the following:<br>1. An environment variable is accessible to the user.<br>2. An environment variable used by the application can be tainted with user supplied data.<br>3. Input data used in an environment variable is not validated properly.<br>4. The variables encapsulation is not done properly. For instance setting a variable as public in a class makes it visible and an attacker may attempt to manipulate that variable. |

As further evidence of the similarity of these two attack patterns, consider Fig. 13. This shows CAPEC-10 and CAPEC-13 within the CAPEC hierarchy of attack patterns. We see that these attack patterns are siblings, i.e., they are specializations of a more general parent pattern (CAPEC-77 *Manipulating User-Controlled Variables*).

The next closest pair of attack patterns in Fig. 12 are CAPEC-114 *Authentication Abuse* and CAPEC-115 *Authentication Bypass*. As shown in Fig. 14, CAPEC-114 and CAPEC-115 are also siblings within the CAPEC hierarchy.

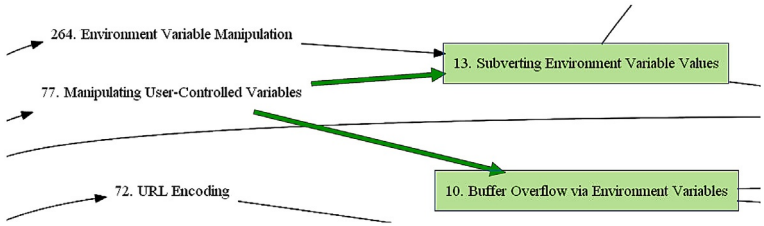


**FIG. 13** CAPEC-10 and CAPEC-13 as siblings in attack pattern hierarchy.

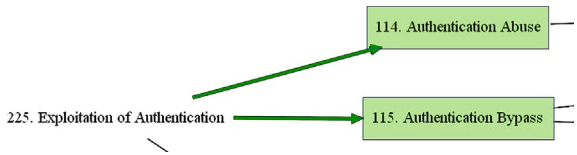


**FIG. 14** CAPEC-114 and CAPEC-115 as siblings in attack pattern hierarchy.

In particular, they are children of the more general attack pattern CAPEC-225 *Exploitation of Authentication*.

Table 8 shows the text content mined for these attack patterns. In comparison to Table 7 (for CAPEC-10 and CAPEC-13), there is considerably more text. In measuring common words for document pairs, one might expect that more text content yields more word matches, causing smaller distances (more similar documents). However, the cosine similarity measures the angle between documents, independent of vector magnitude. That is, the cosine similarity is normalized by the magnitudes of the two document vectors, which are larger for more words. The similarity measures the relative mixture of document terms, not absolute numbers of terms. Another factor in document distances is the inverse document frequency, i.e., more common words have less weight in measuring similarity.

Up to this point, our text mining has been applied to the combined text from these CAPEC attributes: attack pattern name, summary, and attack prerequisites. The resulting clusters are therefore based on general characteristics of attack patterns, along with conditions for their success. However, as described in Section 2, there are a variety of other CAPEC attributes, which characterize attack patterns in different ways. Mining these other attributes can tell us how attack patterns are related in terms of these other characteristics.

As an example, consider Fig. 15. This shows two of the phases (Experiment and Exploit) for the attack execution flow defined for a particular attack pattern (CAPEC-7 *Blind SQL Injection*). Each phase for this attack pattern has two attack steps, with corresponding attack techniques, indicators, outcomes, and security controls.

**TABLE 8** Text Content for Next Two Most Similar Documents of Fig. 12

| ID | Name | Summary | Prerequisites |
|---|---|---|---|
| 114 | Authentication Abuse | An attacker obtains unauthorized access to an application, service or device either through knowledge of the inherent weaknesses of an authentication mechanism, or by exploiting a flaw in the authentication scheme's implementation. In such an attack an authentication mechanism is functioning but a carefully controlled sequence of events causes the mechanism to grant access to the attacker. This attack may exploit assumptions made by the target's authentication procedures, such as assumptions regarding trust relationships or assumptions regarding the generation of secret values. This attack differs from Authentication Bypass attacks in that Authentication Abuse allows the attacker to be certified as a valid user through illegitimate means, while Authentication Bypass allows the user to access protected material without ever being certified as an authenticated user. This attack does not rely on prior sessions established by successfully authenticating users, as relied upon for the "Exploitation of Session Variables, Resource IDs and other Trusted Credentials" attack patterns. | This attack requires the following:<br>1. The application uses environment variables.<br>2. An environment variable exposed to the user is vulnerable to a buffer overflow.<br>3. The vulnerable environment variable uses untrusted data.<br>4. Tainted data used in the environment variables is not properly validated. For instance boundary checking is not done before copying the input data to a buffer. |
| 115 | Authentication Bypass | An attacker gains access to application, service, or device with the privileges of an authorized or privileged user by evading or circumventing an authentication | This attack requires the following:<br>1. An environment variable is accessible to the user. |

*Continued*

**TABLE 8** Text Content for Next Two Most Similar Documents of Fig. 12—Cont'd

| ID | Name | Summary | Prerequisites |
|---|---|---|---|
| | | mechanism. The attacker is therefore able to access protected data without authentication ever having taken place. This refers to an attacker gaining access equivalent to an authenticated user without ever going through an authentication procedure. This is usually the result of the attacker using an unexpected access procedure that does not go through the proper checkpoints where authentication should occur. For example, a web site might assume that all users will click through a given link in order to get to secure material and simply authenticate everyone that clicks the link. However, an attacker might be able to reach secured web content by explicitly entering the path to the content rather than clicking through the authentication link, thereby avoiding the check entirely. This attack pattern differs from other authentication attacks in that attacks of this pattern avoid authentication entirely, rather than faking authentication by exploiting flaws or by stealing credentials from legitimate users. | 2. An environment variable used by the application can be tainted with user supplied data.<br>3. Input data used in an environment variable is not validated properly.<br>4. The variables encapsulation is not done properly. For instance setting a variable as public in a class makes it visible and an attacker may attempt to manipulate that variable. |

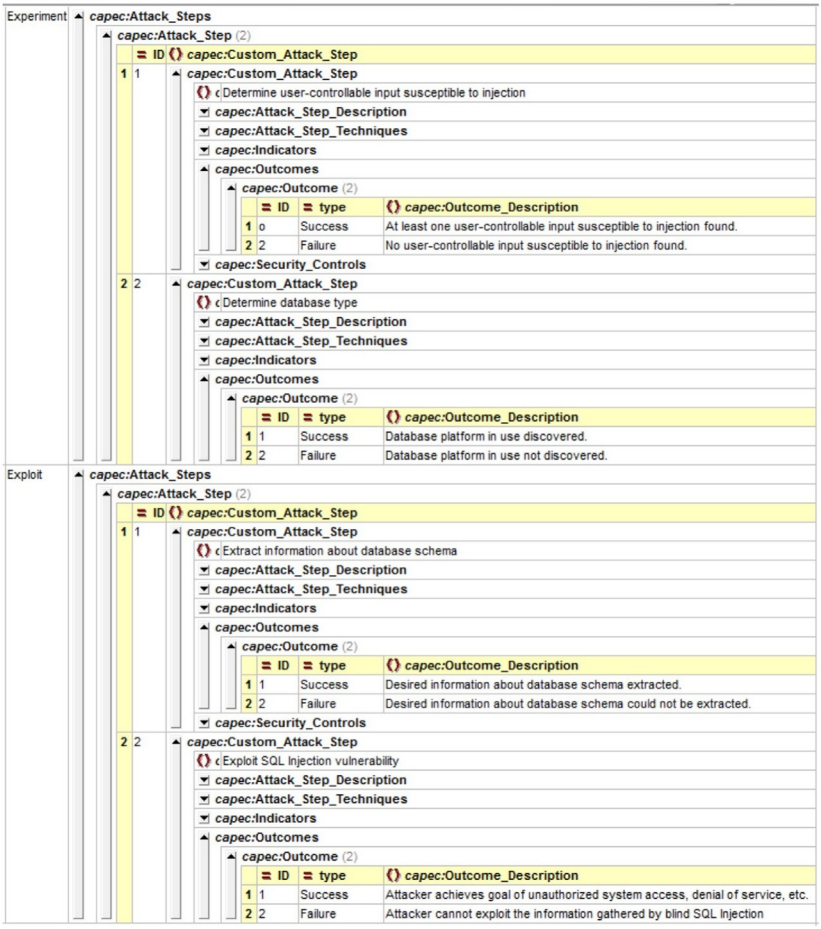


**FIG. 15** Outcomes for attack steps.

In Fig. 15, each attack step describes outcomes in the event of attack success or failure. We can mine this text content to understand commonalities in impact on cyberspace assets. Fig. 16 shows the resulting complete-linkage dendrogram.

The dendrogram in Fig. 16 has 79 attack patterns, which are all those having defined attack step outcomes in CAPEC (as shown in Fig. 15). We select only those outcomes marked as Success, to characterize attack patterns in terms of successful outcomes for the attacker (impact on cyber assets).

In Fig. 16, there are two pairs of attack patterns that have the smallest distance:

- CAPEC-63 *Simple Script Injection* and CAPEC-199 *Cross-Site Scripting Using Alternate Syntax* (Table 9).

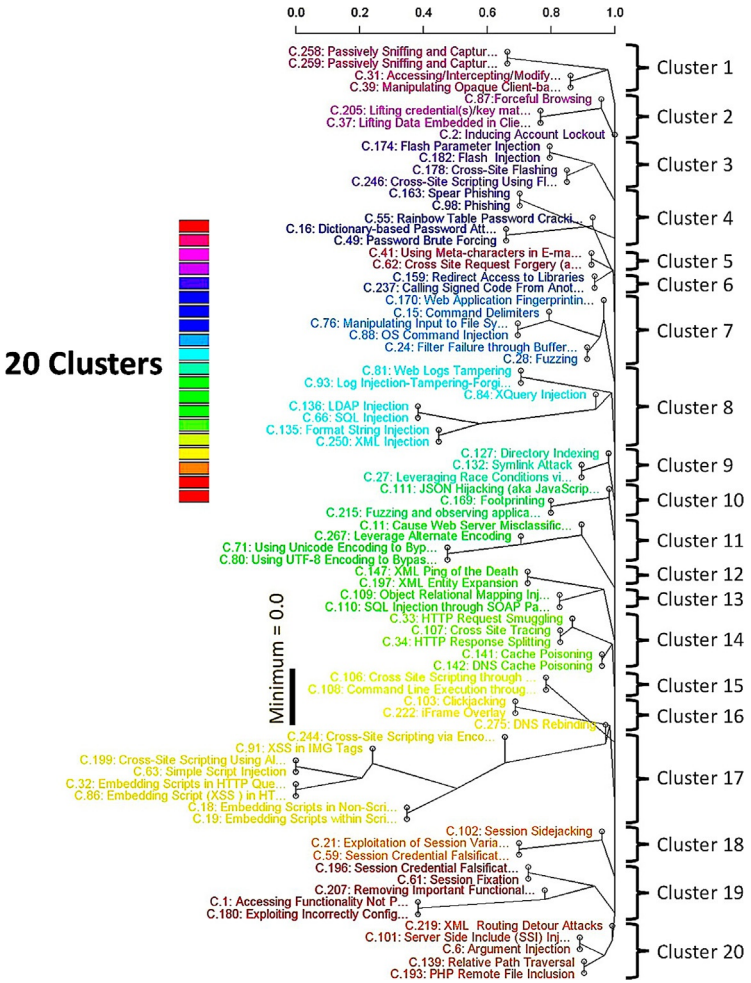


**FIG. 16** Cluster dendrogram based on attack execution outcomes.

- CAPEC-32 *Embedding Scripts in HTTP Query Strings* and CAPEC-86 *Embedding Script (XSS) in HTTP Headers* (Table 10).

As the dendrogram in Fig. 16 shows, the distances for each pair (CAPEC-63/199 and 32/86) are near zero. The pairs then merge into a cluster of four attack patterns at a distance of about 0.2 (i.e., all four are within a distance 0.2 of each other).

Table 9 shows the text mined for CAPEC-63 and CAPEC-199, and Table 10 shows the text mined for CAPEC-32 and CAPEC-86. Careful examination shows that the natural language is nearly identical for all four attack patterns. The key difference is that CAPEC-63 and CAPEC-199 have additional content for the Explore phase:

**TABLE 9** Attack Step Outcomes for CAPEC-63 and CAPEC-199

| ID | Phase | Step | Outcomes |
|---|---|---|---|
| 63 | Explore | Survey the application | Outcomes of this attack:<br>1. A list of URLs, with their corresponding parameters (POST, GET, COOKIE, etc.) is created by the attacker.<br>2. A list of application user interface entry fields is created by the attacker.<br>3. A list of resources accessed by the application is created by the attacker. |
| | Experiment | Probe for XSS vulnerability | Outcomes of this attack:<br>1. The attacker's cross-site scripting string is repeated back verbatim at some point in the web site (if not on the same page). Note that sometimes, the payload might be well encoded in the page, but wouldn't be encoded at all in some other section of the same web page (title, script, etc.) |
| | Exploit | Steal session IDs, credentials, etc. | Outcomes of this attack:<br>1. The attacker gets the user's cookies or other session identifiers.<br>2. The attacker gets the content of the page the user is viewing.<br>3. The attacker causes the user's browser to visit a page with malicious content. |
| | | Forceful browsing | Outcomes of this attack:<br>1. The attacker indirectly controls the user's browser and makes it performing actions exploiting CSRF.<br>2. The attacker manipulates the browser through the steps that he designed in his attack. The user, identified on a website, is now performing actions he is not aware of. |
| | | Content spoofing | Outcomes of this attack:<br>1. The user sees a page containing wrong information. |
| 199 | Explore | Survey the application | Outcomes of this attack:<br>1. A list of URLs, with their corresponding parameters is created by the attacker<br>2. A list of application user interface entry fields is created by the attacker.<br>3. A list of resources accessed by the application is created by the attacker. |

*Continued*

**TABLE 9** Attack Step Outcomes for CAPEC-63 and CAPEC-199—Cont'd

| ID | Phase | Step | Outcomes |
|---|---|---|---|
| | Experiment | Injection payload variations | Outcomes of this attack:<br>1. The attacker's script string is being reflected verbatim at some point in the web site (if not on the same page). Note that sometimes, the payload might be well encoded in the page, but wouldn't be encoded at all in some other section of the same web page (title, etc.) |
| | Exploit | Steal session IDs, credentials, etc. | Outcomes of this attack:<br>1. The attacker gets the user's cookies or other session identifiers.<br>2. The attacker gets the content of the page the user is viewing.<br>3. The attacker causes the user's browser to visit a page with malicious content. |
| | | Forceful browsing | Outcomes of this attack:<br>1. The attacker indirectly controls the user's browser and makes it performing actions exploiting CSRF.<br>2. The attacker manipulates the browser through the steps that he designed in his attack. The user, identified on a website, is now performing actions he is not aware of. |
| | | Content spoofing | Outcomes of this attack:<br>1. The user sees a page containing wrong information. |

- A list of application user interface entry fields is created by the attacker.
- A list of resources accessed by the application is created by the attacker.

This additional content causes the 0.2 difference in distances between CAPEC-63/199 and 32/86 in the Fig. 16 dendrogram.

The closest attack patterns in this dendrogram are part of a cluster of eight CAPEC entries (shaded in Fig. 16):

1. CAPEC-18 *Embedding Scripts in Non-Script Elements*
2. CAPEC-19 *Embedding Scripts within Scripts*
3. CAPEC-32 *Embedding Scripts in HTTP Query Strings*
4. CAPEC-63 *Simple Script Injection*
5. CAPEC-86 *Embedding Script (XSS) in HTTP Headers*
6. CAPEC-91 *XSS in IMG Tags*
7. CAPEC-199 *Cross-Site Scripting Using Alternate Syntax*
8. CAPEC-244 *Cross-Site Scripting via Encoded URI Schemes*

**TABLE 10** Attack Phase Outcomes for CAPEC-32 and CAPEC-86

| ID | Phase | Step | Outcomes |
|---|---|---|---|
| 32 | Explore | Spider | Outcomes of this attack:<br>1. A list of URLs, with their corresponding parameters is created by the attacker. |
| | Experiment | Attempt variations on input parameters | Outcomes of this attack:<br>1. The attacker's cross-site scripting string is repeated back verbatim at some point in the web site (if not on the same page). Note that sometimes, the payload might be well encoded in the page, but wouldn't be encoded at all in some other section of the same web page (title, script, etc.). |
| | Exploit | Steal session IDs, credentials, page content, etc. | Outcomes of this attack:<br>1. The attacker gets the user's cookies or other session identifiers.<br>2. The attacker gets the content of the page the user is viewing.<br>3. The attacker causes the user's browser to visit a page with malicious content. |
| | | Forceful browsing | Outcomes of this attack:<br>1. The attacker indirectly controls the user's browser and makes it performing actions exploiting CSRF.<br>2. The attacker manipulates the browser through the steps that he designed in his attack. The user, identified on a website, is now performing actions he is not aware of. |
| | | Content spoofing | Outcomes of this attack:<br>1. The user sees a page containing wrong information. |
| 86 | Explore | Spider | Outcomes of this attack:<br>1. A list of URLs, with their corresponding HTTP variables is created by the attacker. |
| | Experiment | Probe identified potential entry points for XSS vulnerability | Outcomes of this attack:<br>1. The attacker's cross-site scripting string is repeated back verbatim at some point in the web site (if not on the same page). Note that sometimes, the payload might be well encoded in the page, but would not be encoded at all in some other section of the same web page (title, script, etc.) |

*Continued*

**TABLE 10** Attack Phase Outcomes for CAPEC-32 and CAPEC-86—Cont'd

| ID | Phase | Step | Outcomes |
|---|---|---|---|
| | Exploit | Steal session IDs, credentials, page content, etc. | Outcomes of this attack:<br>1. The attacker gets the user's cookies or other session identifiers.<br>2. The attacker gets the content of the page the user is viewing.<br>3. The attacker causes the user's browser to visit a page with malicious content. |
| | | Forceful browsing | Outcomes of this attack:<br>1. The attacker indirectly controls the user's browser and makes it performing actions exploiting CSRF.<br>2. The attacker manipulates the browser through the steps that he designed in his attack. The user, identified on a website, is now performing actions he is not aware of. |
| | | Content spoofing | Outcomes of this attack:<br>1. The user sees a page containing wrong information. |

These attack patterns are all within a distance of about 0.65 of each other; the next closest pattern to this cluster is almost a distance of one away. As shown in Fig. 17, these attack patterns are also nearby in the CAPEC classification hierarchy. Other nearby attack patterns that are not part of the cluster of eight (e.g., CAPEC-73, 209, 247, 245, 243, and 198) are missing definitions for attack step outcomes.

CAPEC places attack patterns into categories according to particular (subjective) criteria. We can apply our document clustering techniques to the textual content of these categories, to find groups of related categories.

Thus, as for attack patterns, we model each category as a document, with its vector of terms. We then compute distances between each document (attack pattern category) and form clusters. Fig. 18 shows the resulting complete-linkage clustering dendrogram for the CAPEC attack pattern categories. In particular, this uses category name and summary as the natural-language content in our text corpus.

As a check of the textual content of similar attack pattern categories, consider Table 11. This shows the text content for the two closest documents (attack pattern categories) in Fig. 18, i.e., CAPEC-436 *Gain Physical Access* and CAPEC-514 *Physical Security*. We see that the text content is quite similar.

The vector-space model has the advantage of simplicity and is based on established methods of linear algebra. Unlike the Boolean model for

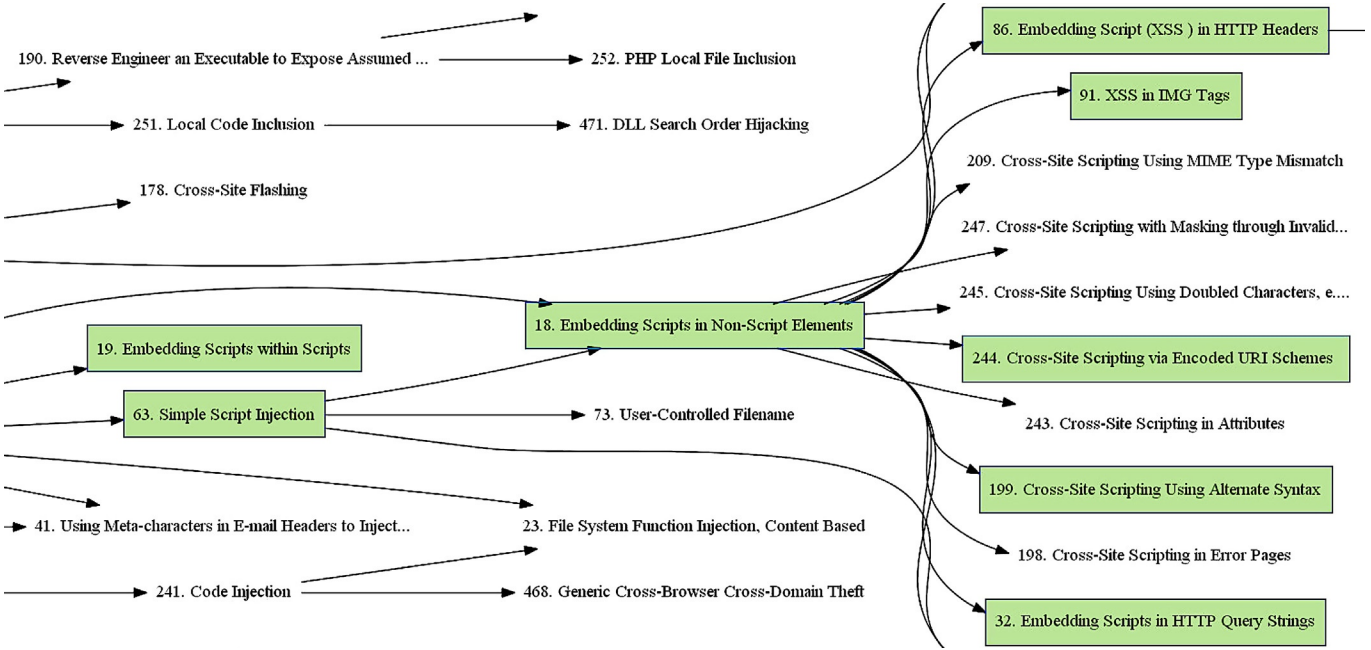


**FIG. 17** Cluster of similar attack patterns within context of CAPEC hierarchy.

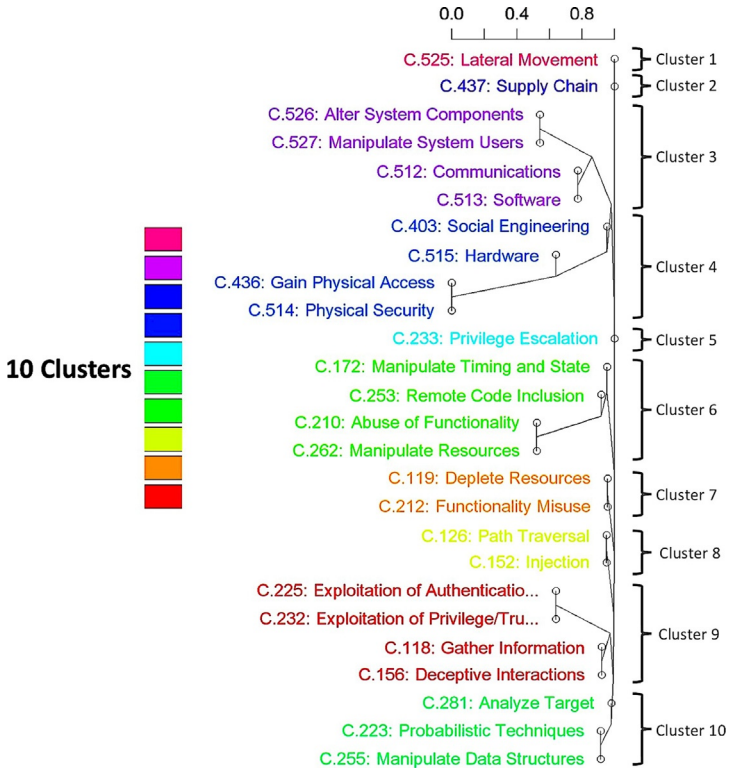


**FIG. 18** Cluster dendrogram for CAPEC attack pattern categories.

**TABLE 11** Text Content for Two Most Similar Attack Pattern Categories of Fig. 18

| ID | Name | Summary |
|---|---|---|
| 436 | Gain Physical Access | Attack patterns within this category focus on gaining physical access to a system or device. The techniques defined by each pattern are used to exploit weaknesses that enable an adversary to achieve physical access. |
| 514 | Physical Security | Attack patterns within this category focus on physical security. The techniques defined by each pattern are used to exploit weaknesses in the physical security of a system in an attempt to achieve a desired negative technical impact. |

information retrieval, it provides a continuous range of document similarities (query-to-document and interdocument), with arbitrary term weightings. This supports partial query matching, document ranking by query relevance, and finding clusters of related documents.

The vector-space model does have some disadvantages. The large number of words in natural language creates a large-dimensional space, although that is less of a problem for CAPEC than for more general corpora. The model presented here does not handle synonymous words, person, tense, number, etc. That is, words must match precisely. There are various ways to extend this basic model (stemming, smoothing, etc.) (Turney and Pantel, 2010), although we do not cover those here. The vector-space model ignores the ordering of terms in a document, and assumes that the terms are statistically independent. While the ability to weight terms provides flexibility, the assignment of weights in practice is more intuitive than formal.

CAPEC organizes attack patterns into hierarchies of parent–child relationships. There are many such possible hierarchies, depending on organizational criteria (CAPEC currently defines 7 of them). With text mining, we are forming groups of related patterns based on analysis of the actual text content, vs human experts grouping them by subjective criteria. Our clustering results show that there are natural clusterings of much of the CAPEC content, (vs a homogenous distribution of attack pattern distances), helping us to understand classes of related attacks. Also, clustering can be a first step in further mining (e.g., classification).

## 5 ATTACK CHAINS

The content in CAPEC can be leveraged as part of more complex security models, such as attack graphs (Jajodia et al., 2011; Noel, 2018; Noel and Jajodia, 2017; Noel et al., 2015a, 2016, 2017) and process models (Noel et al., 2015b). For example, CAPEC attack patterns can include attack execution flows, which describe the steps taken to execute the attack. This is reminiscent of the notion of cyber kill chain (Hutchins et al., 2011), which model the life cycle of attacks in terms of the steps carried out, with potential to break the chain.

Advanced adversaries typically combine multiple attacks to achieve their goals. This extends the kill chain concept to span multiple attack patterns. This section gives examples of how certain attack patterns can be chained together into multistep attacks. While the CAPEC schema includes fields for capturing such requires/provides relationships across attack patterns, those fields are currently not heavily populated in the corpus. In our examples, the multipattern attack chains are created through manual inspection. The techniques described in the previous section provide a foundation for automating this kind of interpattern correlation.

In CAPEC, attack pattern execution flows are divided into three possible phases: Explore, Experiment, and Exploit. Fig. 19 shows the execution flow for a particular CAPEC attack pattern, i.e., CAPEC-170 *Web Application Fingerprinting*. In this case, the attack is for the purposes of gathering information about a target web-based application. It therefore has only the Explore and Experiment phases. Each phase has a number of attack steps, which generally need to be taken in order as one step prepares for another.

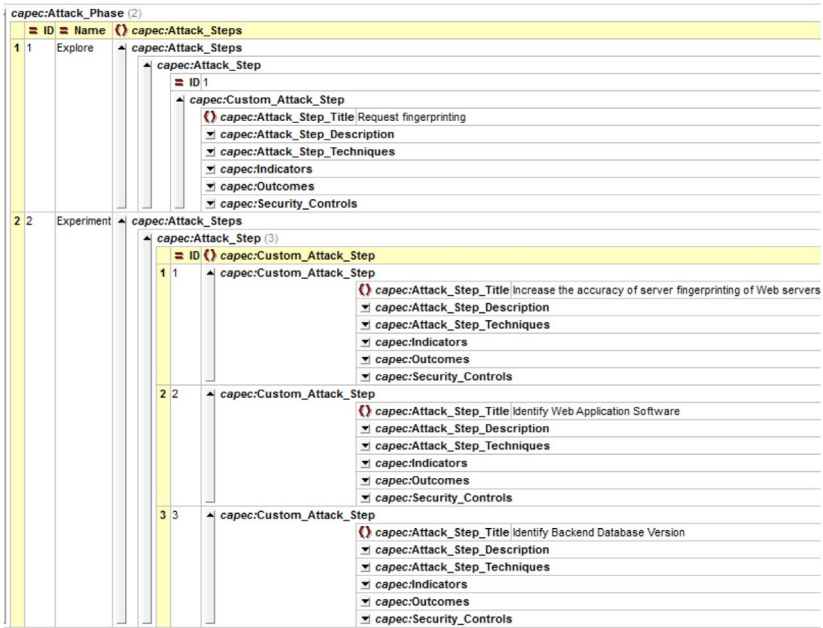


**FIG. 19** Attack phases and execution steps for an attack pattern.

Fig. 20 shows how to model this CAPEC content as sequential steps. This level of modeling abstraction simply shows the attack phases and steps. Other details such as attack techniques, indicators, outcomes, and security controls could be modeled as a separate decomposition of each step.

The modeled execution flow in Fig. 20 includes notional levels of likelihood and difficulty, which can be applied to model analysis. For example, in process models, we can populate attack time durations and probabilities of attack success for Monte Carlo simulations. Similarly, we can populate attack graph models with likelihood of success. Fig. 20 also includes the attacker and victim machines for instantiated attacks on a particular network. This captures the kind of step-wise attack progression modeled in network attack graphs. It can also be leveraged for determining attack effects on cyberspace assets, e.g., for mission impact (Musman et al., 2011; Turner and Musman, 2018).

As the notions of attack execution flows and kill chains suggest, cyber attacks generally follow multiple stages, where success in one stage (attack pattern) provides new attack capabilities for subsequent stages (attack patterns). This is illustrated in Fig. 21.

The purpose of CAPEC-170 *Web Application Fingerprinting* is to identify the type of database that is behind a web-based front-end application. This

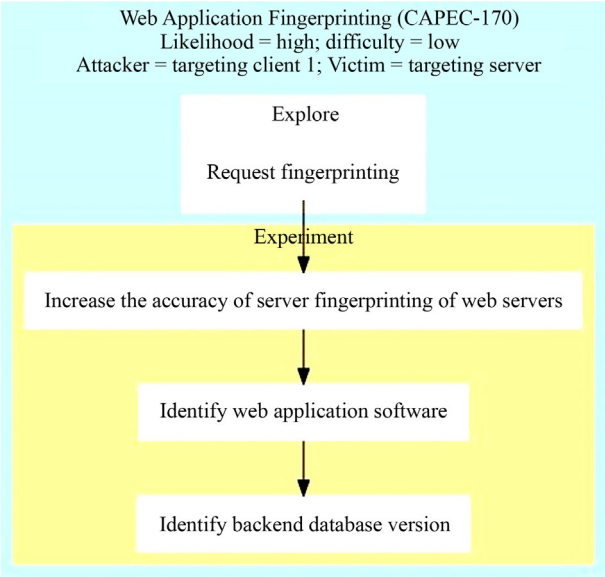


**FIG. 20** Attack execution flow for CAPEC-170 Web Application Fingerprinting.

provides key information needed to launch a subsequent attack against the database. In Fig. 21, CAPEC-170 provides the attacker with enough information about the database to understand that it communicates through Simple Object Access Protocol (SOAP), and is potentially vulnerable to a SOAP-based attack. Then CAPEC-110 *SQL Injection through SOAP Parameter Tampering* represents a particular kind of SOAP-based attack. The final outcome of CAPEC-110 is that the attacker can inject arbitrary SQL commands, e.g., to read and/or modify the database.

We can extend this model by considering other related attack patterns within CAPEC, as shown in Fig. 22. In this model, CAPEC-170 *Web Application Fingerprinting* provides information about the target database for additional attacks. In particular, there are three attack patterns that depend on specific database vulnerabilities:

1. CAPEC-110 *SQL Injection through SOAP Parameter Tampering*
2. CAPEC-15 *Command Delimiters*
3. CAPEC-109 *Object Relational Mapping Injection*

Fig. 22 includes a choice of two possible outcomes from CAPEC-170: either the attacker is able to understand the database well enough to launch one of these specific attacks, or not. If not, then there is one other attack option available: CAPEC-7 *Blind SQL Injection*, which is more of an exploratory approach to attacking a database.

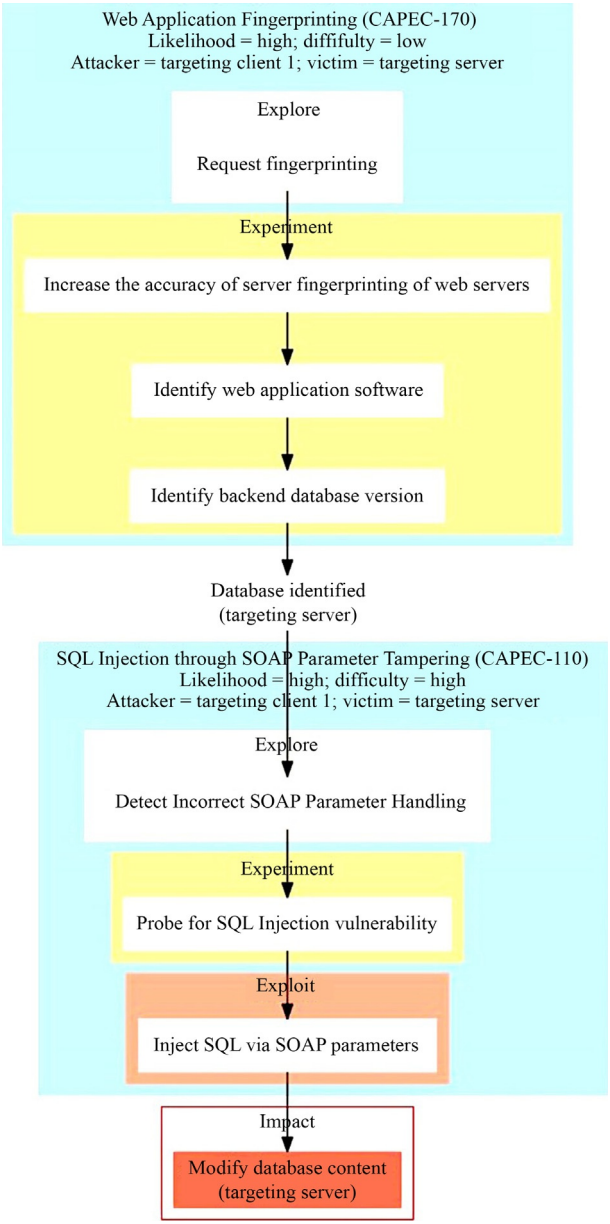


**FIG. 21** Chaining multiple attack patterns.

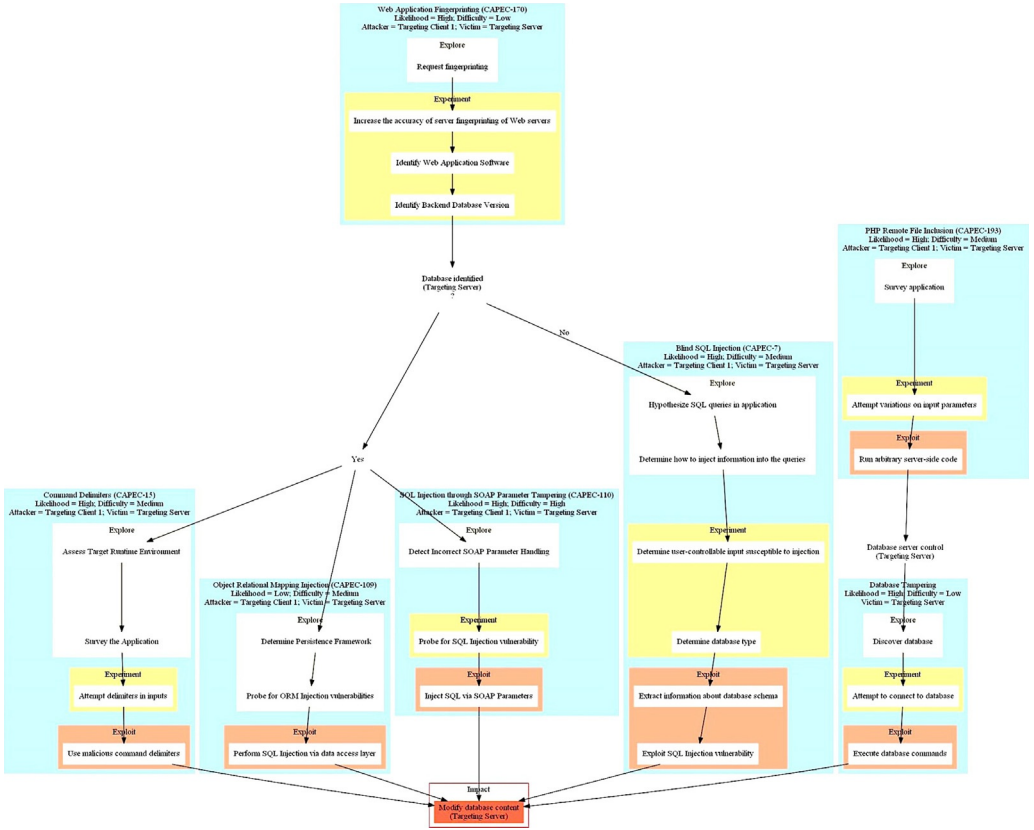


**FIG. 22** More complex combinations of attack patterns.

## 6 ATTACK PATTERN HIERARCHIES

In CAPEC, attack patterns are organized hierarchically from general to specific, providing a range of pattern abstraction levels. Understanding the taxonomic relationships (e.g., for selecting an appropriate level of detail for modeling and analysis) requires effective navigation of this hierarchical structure. This section describes interactive tree visualization of the CAPEC hierarchy, as a more effective alternative to simply following hyperlinks in textual descriptions of attack patterns.

There are many possible ways of defining attack pattern hierarchies. Rather than defining a single strict taxonomy, CAPEC supports more flexible definitions of hierarchical relationships, captured through views. Views define the aspect in which attack patterns are deconstructed into more specialized ones.

To leverage techniques for tree visualization, we transform the partially ordered CAPEC view graph to a corresponding tree. CAPEC views are generally quite tree-like already. Still, in our transformation from acyclic (partial ordered) graph to tree, we retain all the original parent–child relationships. We then explore a number of interactive web-based tree visualizations to better understand the CAPEC hierarchy.

To cast a CAPEC view graph as a tree, we replicate attack patterns that have multiple parents. This eliminates multiple inheritance (specialization), including inheritance from common ancestors (e.g., parent and grandparent). Still, we preserve all existing edges (parent–child relationships) of the original graph. While our treatment is top down to illustrate one possible graph analysis of CAPEC, the visual approaches we describe can be applied more generally, e.g., for a bottom-up analysis of hierarchical relationships above a particular attack pattern.

A CAPEC view defines a graph of parent–child relationships, with each child a specialization of its parent. Vertices are attack patterns or attack pattern categories, each with a unique standard CAPEC identifier. An edge points from a parent (category or pattern) to child (pattern). For the default view (CAPEC-1000 *Mechanism of Attack*), the graph is 465 vertices (451 attack patterns and 14 categories) and 507 edges (parent–child relationships).

A CAPEC view is a directed acyclic graph (a partial ordering of parents to children). For visualization, it is convenient to cast the partial order as a tree. To do this, we start from each root (vertex that has no parent), and find the full hierarchy of children beneath it. Some of the resulting hierarchies are acyclic graphs rather than trees, such as in Fig. 23. This important point in this figure is the structure of CAPEC hierarchical relationships, e.g., certain attack patterns with multiple parents (node labels are immaterial).

To transform the partially ordered portion of the CAPEC graph to a corresponding set of trees (a forest), we consider the graph from the top down. That is, we start from each root (i.e., vertex that has no parent), and find the full hierarchy of children beneath it. In cases in which portions of the graph

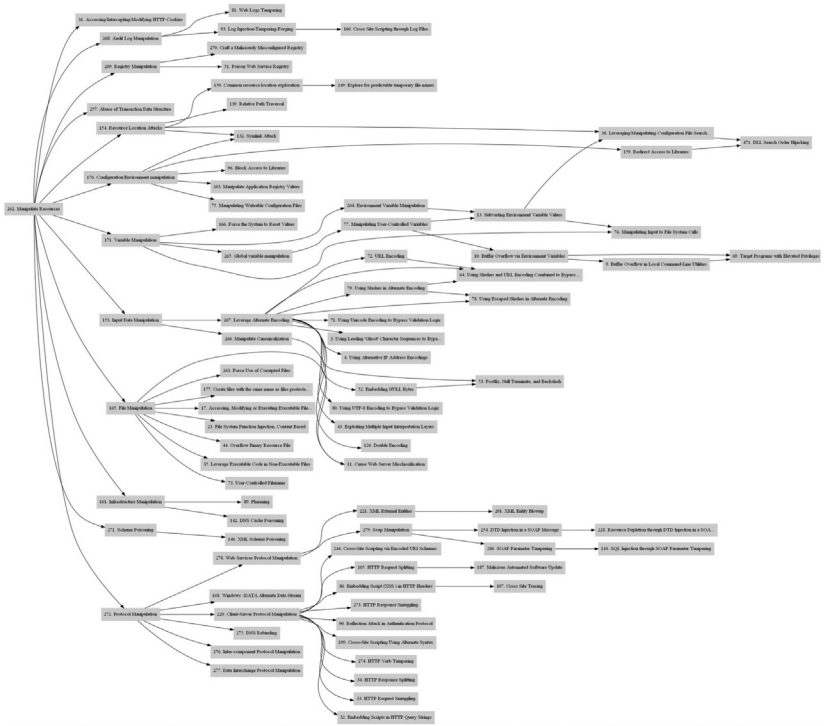

**FIG. 23** Example CAPEC hierarchy (rooted at CAPEC-262 *Manipulate Resources*).

fall beneath multiple roots, those subgraphs are represented in each top-down hierarchy (i.e., replicated).

The result is 30 distinct hierarchies, where a hierarchy is a root vertex and its full subgraph of descendants. Fig. 24 shows the 15 largest hierarchies, labeled by root and sorted by hierarchy size (number of descendants). In the case of replicated attack patterns (those under multiple hierarchies), they are counted for each hierarchy.

Among the 30 CAPEC hierarchies that we construct in this way, 25 are rooted trees. The remaining five hierarchies are acyclic graphs (partial orders) with a single root. These partial orders are actually quite tree-like. In fact, of the 459 vertices in the CAPEC graph, 395 of them (86%) have a single parent as required for a tree.

Taxonomies are hierarchical classification structures, typically organized by generalization–specialization relationships (Vateekul, 2012). As such, they must be trees (vs more general partial orders). CAPEC is not intended to be a strict taxonomy, but rather a general (partial order) graph. Its primary intent is not to classify attack patterns. Rather, it is designed for informative exploration and identification of relevant patterns for particular use cases and contexts.

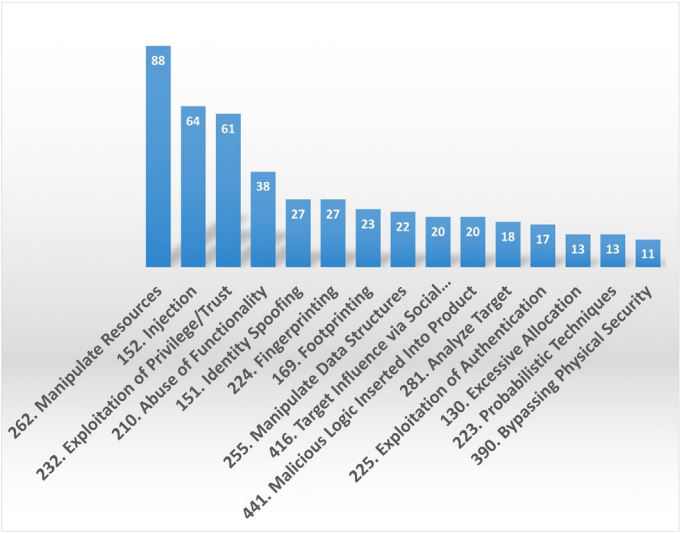


**FIG. 24** The 15 largest CAPEC hierarchies (among 30 total).

To cast the partial-order hierarchy of a CAPEC view as a tree, we apply the same approach of working top down from the hierarchy root, and replicating vertices (attack patterns) that have multiple parents. Fig. 25 shows an example of this. Note that this is a subgraph of one of the full hierarchies; the remainder of the graph is omitted for clarity.

Algorithmically, we start at the root of a hierarchy, and recursively process its children. At each encountered vertex, we add the vertex to the output tree. Then, we recursively call the same function for the current node's children. Rather than marking each visited vertex and avoiding marked ones (as usual in graph traversal), we replicate vertices (attack patterns) each time they occur as a child. In this way, all parent–child relationships are retained. Because the number of vertices with multiple parents is relatively low, we are not introducing excessive numbers of replicated attack patterns.

This transformation from partial order (directed acyclic graph) to tree eliminates inheritance (specialization) from multiple parents. This includes the special case of inheritance from common ancestors (e.g., parent and grandparent). Like transitive reduction (Aho et al., 1972), this operation prevents edges that can be inferred through transitivity. However, unlike transitive reduction, we preserve all edges (parent–child relationships) of the original graph.

Functionally, vertices with multiple parents are represented with clone instances for each parent. This occurs recursively, so that cloned parents each have their own cloned subtrees. In Fig. 25, this happens in the case of CAPEC-9 *Buffer Overflow in Local Command-Line Utilities*. Its subtree CAPEC-69 *Target Programs with Elevated Privileges* is cloned twice (once

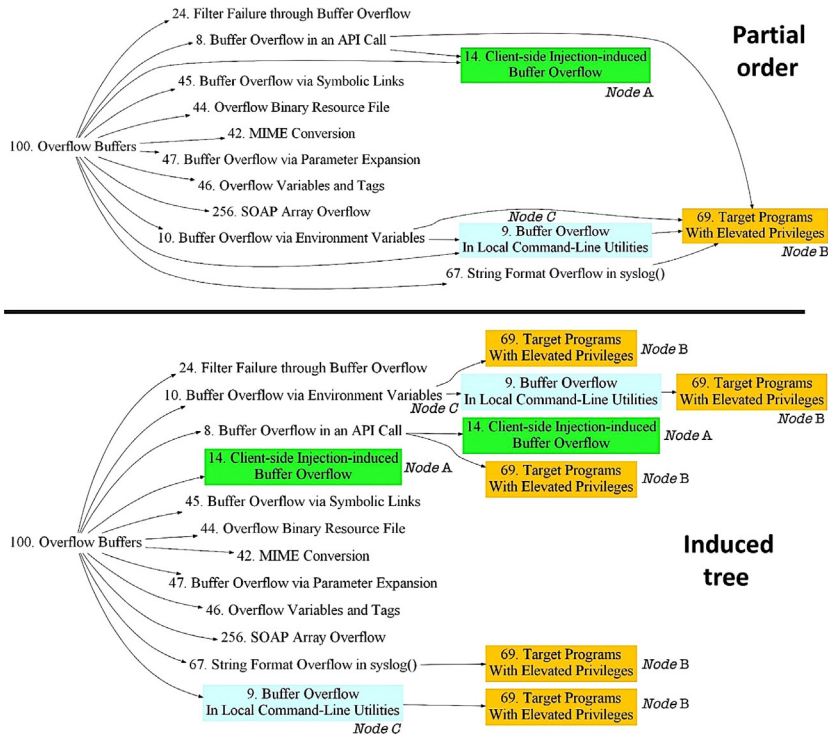


**FIG. 25** Inducing trees on attack pattern partial orders.

for each parent of CAPEC-9). Then, CAPEC-69 is cloned again for its each of its other parents (other than CAPEC-9).

In this way, we transform the directed acyclic CAPEC graph to a set of trees (a forest). To build a single unified tree, we simply create an overall CAPEC root vertex, and make it the parent of each tree in the induced forest. Fig. 26 shows the resulting full taxonomic tree.

A simple way to visualize a CAPEC parent–child tree is a vertex-edge diagram with vertices in the same level of the tree aligned along parallel lines (Reingold and Tilford, 1981). Fig. 27 shows an example of this. In this visualization, each circle is a CAPEC category or attack pattern. The shaded circles are subtrees collapsed into a single vertex. Clicking on a shaded circle expands the next level of the subtree. This is shown on the top of Fig. 27, for the case of the *Injection* category.

A weakness of the Cartesian tree visualization is that its layout is not the most efficient use of the display space, e.g., for showing the entire tree structure as in the bottom of Fig. 27. One way to pack the tree display into a smaller space is to employ a radial layout, a classic design pattern for diagramming hierarchies (Schulz et al., 2011). Fig. 28 is a radial visualization for the CAPEC parent–child tree.

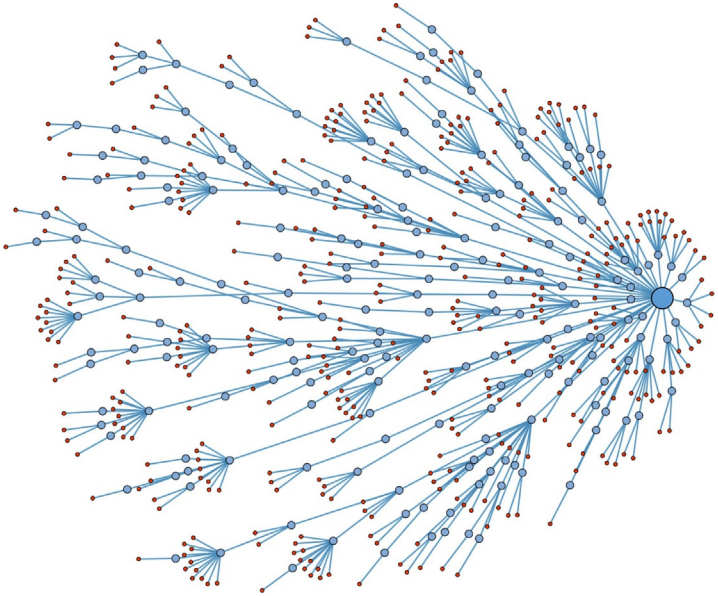

**FIG. 26** The complete CAPEC taxonomy as a tree.

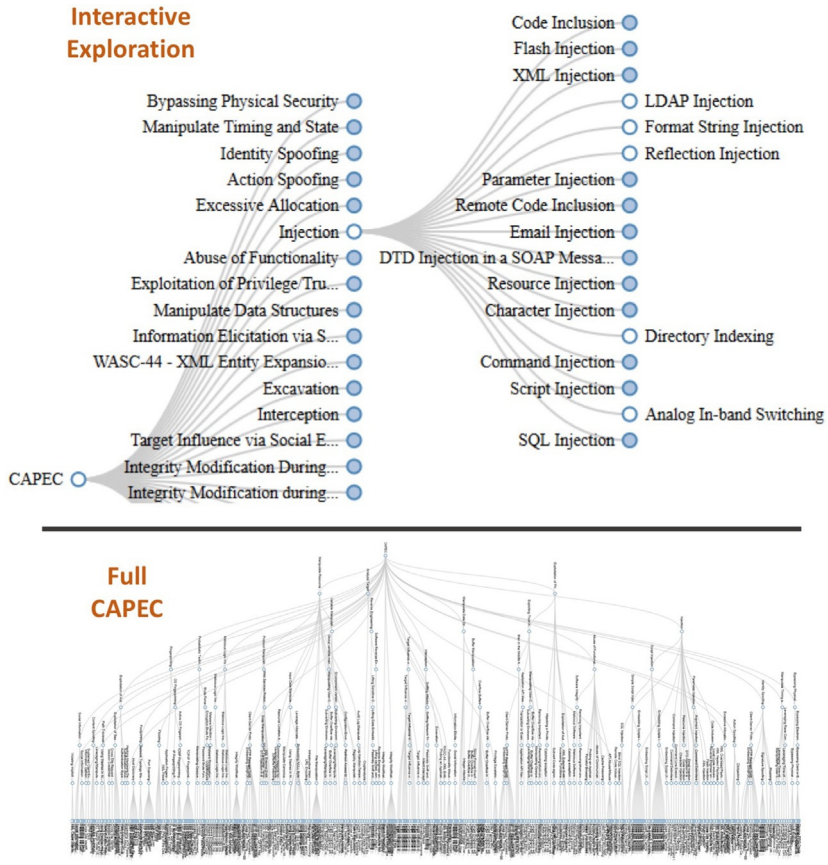


**FIG. 27** Cartesian visualization of CAPEC taxonomy.

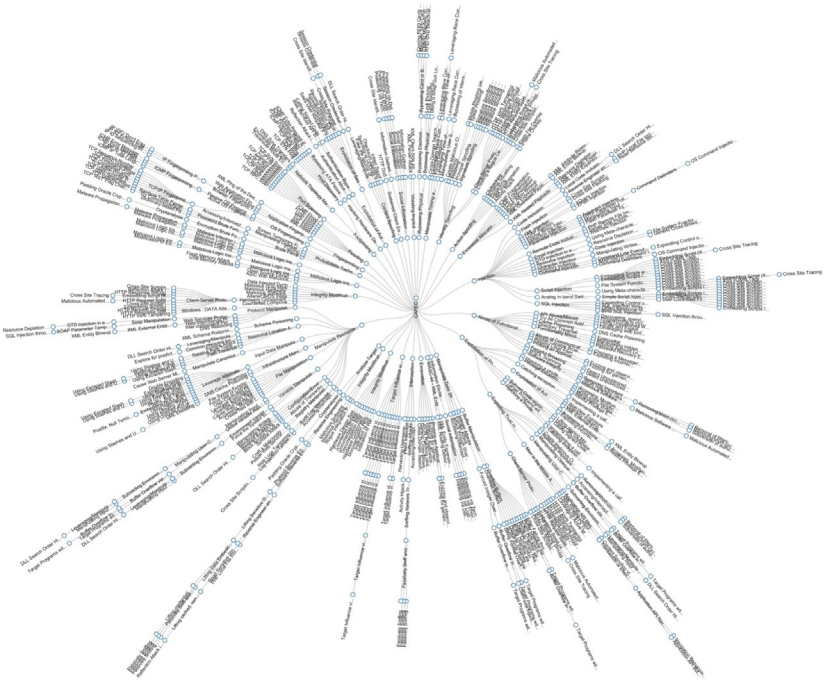

**FIG. 28** Radial visualization of CAPEC taxonomy.

While the visualization in Fig. 28 shows the entire CAPEC tree in a single view, it is too cluttered for easy comprehension. There is a class of tree visualization techniques that represents parent–child relationships implicitly (Schulz, 2011), rather than explicitly drawing vertices and edges. One such technique is the sunburst visualization (Stasko and Zhang, 2000), which draws radial wedges for each level of the tree.

There is a particular kind of tree visualization that is specifically designed to fill the display space completely, known as treemaps (Johnson and Shneiderman, 1991). The rectangular treemap recursively splits the display into rectangles for each tree vertex at a level, alternating horizontally and vertically for each level.

However, rectangular treemaps can be hard to interpret, especially for larger more complex trees (Balzer et al., 2005; Bruls et al., 2000). Because the bounding rectangles have parallel sides at all levels, it can be difficult to distinguish levels. Also, aspect ratios can be very different for rectangles of equal area, making visual comparisons of size more difficult. An alternative is circular treemaps (Wetzel, 2003), which uses nested circles instead of rectangles. While circular treemaps do not fill the display space completely, their multilevel nesting (tree) structure is easy to recognize. Also, the shape (aspect ratio) is the same for all vertices.

A more recent approach is treemaps that recursively divide the display space via centroidal Voronoi tessellation (Balzer et al., 2005). These have the advantages of near-unity aspect ratio and stable interactive dynamics, e.g., when zooming (Sud et al., 2010). Voronoi tessellation divides space into regions (cells) according to a predefined set of points (generators). A cell contains all points closer to its generator than to any other. In our case, each cell and its generator correspond to a vertex of the tree to visualize.

In the centroidal form, each generator is in the mean position (centroid) of its cell. Minimizing the overall energy of a tessellation corresponds to minimizing the aspect ratio of its cells. Finding a centroidal Voronoi tessellation with global minimum energy is an NP-complete problem (Du et al., 1999), but we can compute noncentroidal tessellations efficiently to approximate the centroidal case.

The approximation method starts with the generators randomly distributed in the display space, and then iteratively samples them, moving them to their centroid position and recomputing the tessellation. The process terminates when the overall energy reaches a given error threshold. The model can be extended by weighting each generator, so that the area of its cell is increased.

Fig. 29 is a Voronoi treemap visualization for the CAPEC parent–child tree. Here, the cells at each level are weighted by their number of children (attack patterns), so that patterns with more children have more display area. There is also a bias to place larger cells near the center of the display.

The top of Fig. 29 shows the full CAPEC tree. Label occlusion is addressed by making the lower-level tessellations semitransparent. Also, label size (larger labels for higher tree levels) emphasizes the currently viewed level. If we click on a cell in the interactive visualization of Fig. 29, it zooms in to show that subtree. This is shown in the bottom of Fig. 29, for the Injection category. This visualization shows neighboring parts of the tree under zoom, helping to maintain overall context.

## 7 ANALYTIC ENVIRONMENT

In previous sections, we explore various options for leveraging CAPEC for attack modeling. This section describes our data and software environment for this analysis.

The bulk CAPEC content is distributed as XML. The structure of this XML is described through XML schema. From the XML schema, we generate code that corresponds to the XML data structure of CAPEC. The generated code then parses the CAPEC XML and populates data objects in our code.

Our code for analyzing CAPEC is written in Java. We leverage Java Architecture for XML Binding (JAXB) (Oracle, 2013) to bind the CAPEC XML to Java objects. XML schema bindings are also available in other programming languages, such as PyXB for Python (Sourceforge, 2017). The XJC binding compiler (Oracle, 2018) generates Java classes that are a

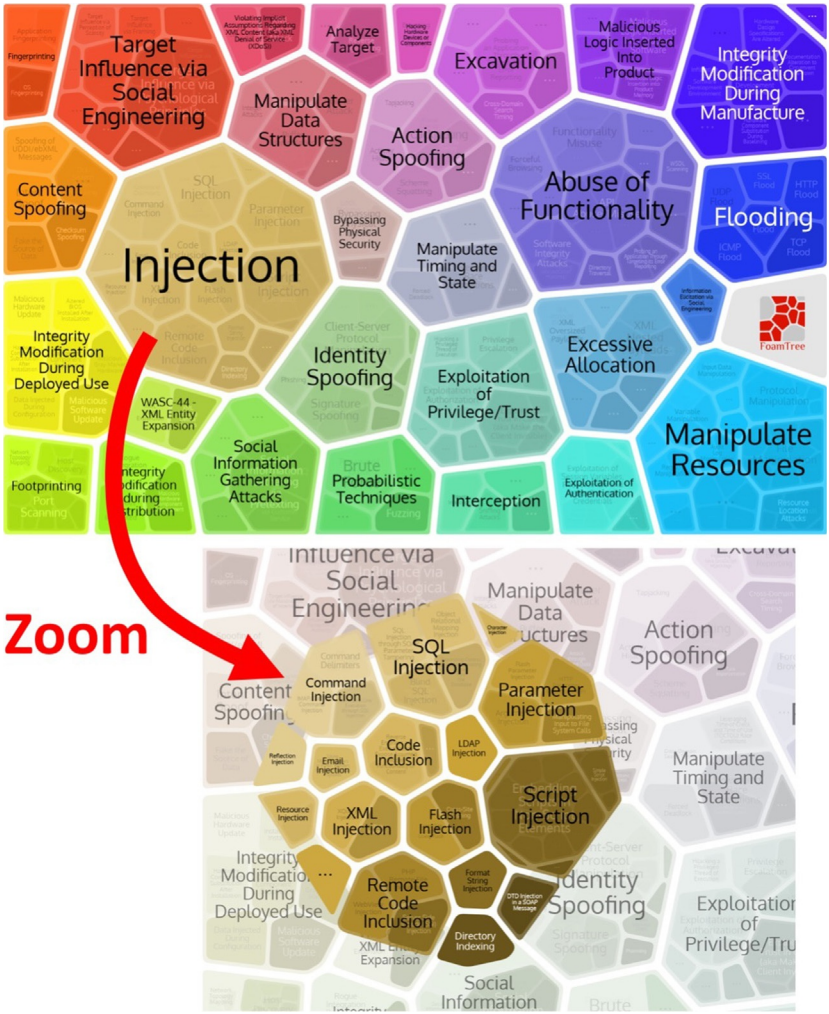


**FIG. 29** Voronoi treemap of CAPEC taxonomy.

reflection of the CAPEC XML schema. The JAXB interface then unmarshals the XML, i.e., parses the XML string and maps its elements to corresponding Java objects, according to the schema. This automates the entire process of populating a data structure with the CAPEC content. We then build our application-specific code that analyzes the CAPEC content.

For our analysis of the natural-language content of CAPEC, we employ custom Java code for building document vectors and computing distances between documents or from documents to query vectors. For the clustering analysis and dendrogram visualizations, we leverage the R language (The R Project for

Statistical Computing, 2018). In particular, we generate a document distance matrix via Java, then write it to file as comma-separated values (CSV). We then import the CSV distance matrix to R.

Our analysis includes building a graph of hierarchical relationships among the CAPEC attack patterns. For this we leverage the GraphStream library (Dutot et al., 2007) for graph modeling. This allows us to simply iterate through the attack patterns in the CAPEC content objects, and instantiate graph vertices and edges for each parent–child relationship in each pattern. We can visualize our CAPEC hierarchy in GraphStream directly, or apply other visualization libraries to leverage their visual capabilities for particular use cases. For hierarchical layouts of general directed graphs (Gansner et al., 1993), we can apply GraphViz (Gansner and North, 2000).

CyGraph (Noel et al., 2016) provides a layered graph model with interactive graph state changes. The D3.js JavaScript library (Bostock, 2017) provides interactive browser-based visualization through Data-Driven Documents (D3) (Bostock et al., 2011). The FoamTree JavaScript library (Carrot Search, 2018) implements Voronoi treemap visualizations. The input data for CyGraph, D3.js, and FoamTree are expressed as JavaScript Object Notation (JSON). Like XML, JSON is a serialized string representation of a nested (tree) data structure. In our case, the JSON represents the CAPEC taxonomic tree that we build. To serialize our Java objects to JSON string, we leverage the Google Gson library (Google, 2018). In this framework, one populates a Java object with the structure and content of the JSON to be serialized. Gson then uses Java reflection (Oracle, 2017) to examine the details of the object and perform serialization.

## 8 SUMMARY

This chapter examines how natural-language processing can be applied to the CAPEC attack catalog for cybersecurity analytics. CAPEC is a standardized corpus of cyberattack patterns, which can be leveraged for understanding the scope and limitations of security capabilities. CAPEC captures knowledge about classes of attacks, including details about attack phases, vulnerable attack surface, attacker resources required, and mitigations.

We describe the general structure and content of CAPEC attack patterns, such as the technical challenges that the attacker must overcome, attack impacts, and recommendations for defense (underlying weaknesses, solutions, attack warnings, etc.). We also provide some example applications of CAPEC for cyberattack modeling, i.e., analyzing the corpus to find attack patterns of relevance to particular attack scenarios.

For mining the natural-language content of CAPEC we apply a vector-space approach. This models CAPEC attack patterns as documents, for which we build vectors of terms. We define vector-based distances that measure similarity between documents, or distances of documents to query term vectors. We apply these distances to relevance ranking of query matches and perform

cluster analysis to discover groups of related attack patterns, e.g., for expanding potential patterns of interest.

We apply tree visualization techniques for analyzing the hierarchical structure of attack pattern clustering. We also examine the taxonomic structure of CAPEC attack patterns, organized as classes of attacks from general to specific. We cast the CAPEC taxonomy from a directed acyclic graph to a tree and demonstrate interactive visualizations for exploring it. Overall, these techniques provide a range of capabilities for building analytic models from a shared corpus of cybersecurity information.

## REFERENCES


Aho, A., Garey, M., Ullman, J., 1972. The transitive reduction of a directed graph. SIAM J. Comput. 1 (2), 131–137.

Balzer, M., Deussen, O., Lewerentz, C., 2005. Voronoi treemaps for the visualization of software metrics. In: ACM Symposium on Software Visualization.

Barnum, S., Sethi, A., 2007. Attack patterns as a knowledge resource for building secure software. In: Object Management Group First Software Assurance Workshop.

Bostock, M., Ogievetsky, V., Heer, J., 2011. D3: data-driven documents. IEEE Trans. Vis. Comput. Graph. 17 (12), 2301–2309.

Bostock, M., 2017. D3.js Data-Driven Documents. http://d3js.org/.

Bruls, M., Huizing, K., Van Wijk, J., 2000. Squarifed treemaps. In: Joint Eurographics and IEEE TCVG Symposium on Visualization.

Carrot Search, 2018. FoamTree: Interactive Voronoi Treemaps. http://carrotsearch.com/foamtree-overview.

Chickowski, E., 2017. The Rising Tide of Crimeware-as-a-Service. https://www.darkreading.com/threat-intelligence/the-rising-tide-of-crimeware-as-a-service/d/d-id/1329102.

Denisov, S., 1997. Fractal binary sequences: Tsallis thermodynamics and the Zipf law. Phys. Lett. A 235, 447–451.

Du, Q., Faber, V., Gunzburge, M., 1999. Centroidal Voronoi tessellations: applications and algorithms. SIAM Rev. 41 (4), 637–676.

Dutot, A., Guinand, F., Olivier, D., Pigné, Y., 2007. Graphstream: a tool for bridging the gap between complex systems and dynamic graphs. In: Emergent Properties in Natural and Artificial Complex Systems, Dresden, Germany.

Frazier, G., Duong, Q., Wellman, M., Petersen, E., 2011. Incentivizing responsible networking via introduction-based routing. In: McCune, J.M., Balacheff, B., Perrig, A., Sadeghi, A.R., Sasse, A., Beres, Y. (Eds.), Trust and Trustworthy Computing, Lecture Notes in Computer Science, 6740, Springer, pp. 277–293.

Gansner, E., North, S., 2000. An open graph visualization system and its applications to software engineering. Softw. Pract. Exp. 30 (11), 1203–1233.

Gansner, E., Koutsofios, E., North, S., Vo, K.P., 1993. A technique for drawing directed graphs. IEEE Trans. Softw. Eng. 19 (3), 214–230.

Google, 2018. Gson A Java Library to Convert JSON to Java Objects and Vice-Versa. http://code.google.com/p/google-gson/.

Hoglund, G., McGraw, G., 2004. Exploiting Software: How to Break Code. Addison-Wesley.

Hutchins, E., Cloppert, M., Amin, R., 2011. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. In: Annual International Conference on Information Warfare and Security, Washington, DC.

Jajodia, S., Noel, S., Kalapa, P., O'Berry, B., Jacobs, M., Robertson, E., Weierbach, R., 2011. Network Attack Modeling, Analysis, and Response, US Patent 7,904,962, 8 March. .

Johnson, B., Shneiderman, B., 1991. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: IEEE Conference on Visualization, San Diego, CA.

Lee, D., Chuang, H., Seamons, K., 1997. Document ranking and the vector-space model. IEEE Softw. 14 (2), 67–75.

Mandelbrot, B., 1966. Information Theory and Psycholinguistics: A Theory of Words Frequencies. MIT Press.

Martin, R., 2009. Making security measurable and manageable. Crosstalk: J. Defense Softw. Eng. 26–32 (September/October).

Montemurro, M., 2001. Beyond the Zipf-Mandelbrot law in quantitative linguistics. Physica A 300, 567–578.

Moore, A., Ellison, R., Linger, R., 2001. Attack Modeling for Information Security and Survivability, CMU/SEI-2001-TN-001. .

Musman, S., Tanner, M., Temin, A., Elsaesser, E., Loren, L., 2011. Computing the impact of cyber attacks on complex missions. In: IEEE International Systems Conference.

Noel, S., 2018. A Review of Graph Approaches to Network Security Analytics. Springer.

Noel, S., Jajodia, S., 2017. Measuring Enterprise Cybersecurity Risk Through Topological Vulnerability Analysis. In: Springer.

Noel, S., Harley, E., Tam, K.H., Gyor, G., 2015a. Big-data architecture for cyber attack graphs: representing security relationships in NoSQL graph databases. In: IEEE Symposium on Technologies for Homeland Security (HST), Boston, MA.

Noel, S., Ludwig, J., Jain, P., Johnson, D., Thomas, R., McFarland, J., King, B., Webster, S., Tello, B., 2015b. Analyzing mission impacts of cyber actions (AMICA). In: NATO IST-128 Workshop on Cyber Attack Detection, Forensics and Attribution for Assessment of Mission Impact, Istanbul, Turkey.

Noel, S., Harley, E., Tam, K.H., Limiero, M., Share, M., 2016. CyGraph: Graph-Based Analytics and Visualization for Cybersecurity. In: Elsevier.

Noel, S., Bodeau, D., McQuaid, R., 2017. Big-data graph knowledge bases for cyber resilience. In: NATO IST-153 Workshop on Cyber Resilience, Munich, Germany.

Oracle, 2013. Java Architecture for XML Binding (JAXB). http://www.oracle.com/technetwork/articles/javase/index-140168.html.

Oracle, 2017. Trail: The Reflection API (The Java Tutorials). http://docs.oracle.com/javase/tutorial/reflect/.

Oracle, 2018. Java Architecture for XML Binding Compiler (xjc). https://docs.oracle.com/javase/7/docs/technotes/tools/share/xjc.html.

Reingold, E., Tilford, J., 1981. Tidier drawings of trees. IEEE Trans. Softw. Eng. 7 (2), 223–228.

The R Project for Statistical Computing, 2018. http://www.r-project.org/.

Robertson, S., 2004. Understanding inverse document frequency: on theoretical arguments for IDF. J. Doc. 60 (5), 503–520.

Salton, G., 1983. Introduction to Modern Information Retrieval. McGraw-Hill.

Salton, G., Buckley, C., 1996. Term weighting approaches in automatic text retrieval. Inf. Process. Manag. 32 (4), 431–443.

Schulz, H., 2011. Treevis.net: a tree visualization reference. IEEE Comput. Graph. Appl. 31 (6), 11–15.

Schulz, H., Hadlak, S., Schumann, H., 2011. The design space of implicit hierarchy visualization: a survey. IEEE Trans. Vis. Comput. Graph. 17 (4), 393–411.

Sourceforge, 2017. PyXB: Python XML Schema Bindings. http://pyxb.sourceforge.net/.

Stasko, J., Zhang, E., 2000. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In: IEEE Symposium on Information Visualization, Salt Lake City, Utah.

Sud, A., Fisher, D., Lee, H.P., 2010. Fast dynamic Voronoi treemaps. In: International Symposium on Voronoi Diagrams in Science and Engineering, Saint Petersburg, Russia.

The MITRE Corporation, 2013. Making Security Measurable. http://makingsecuritymeasurable.mitre.org/.

The MITRE Corporation, 2014. Directory of Efforts by Organization Name. http://makingsecuritymeasurable.mitre.org/directory/organizations/.

The MITRE Corporation, 2018a. The Common Attack Pattern Enumeration and Classification - A Community Resource for Identifying and Understanding Attacks. https://capec.mitre.org/.

The MITRE Corporation, 2018b. CVE - Common Vulnerabilities and Exposures. https://cve.mitre.org/.

The MITRE Corporation, 2018c. CWE - Common Weaknesses Enumeration. http://cwe.mitre.org/.

Turner, A., Musman, S., 2018. Applying the cybersecurity game to a point-of-sale system. In: Disciplinary Convergence in Systems Engineering ResearchSpringer, pp. 29–144.

Turney, P., Pantel, P., 2010. From frequency to meaning vector space models of semantics. J. Artif. Intell. Res. 37, 141–188.

United States Computer Emergency Response Team (US-CERT), 2013. Understanding Denial-of-Service Attacks. In: US-CERT Security Tip (ST04-015).

Vateekul, P., 2012. Hierarchical Multi-Level Classification: Going beyond Generalization Trees (doctoral thesis), University of Miami.

Venables, W., Ripley, B., 1994. Modern Applied Statistics With S-Plus. Springer-Verlag.

Wetzel, K., 2003. Pebbles Using Circular Treemaps to Visualize Disk Usage, Sourceforge Project. http://lip.sourceforge.net/ctreemap.html.

Wong, S., Ziarko, W., Raghavan, V., Wong, P., 1987. On modeling of information retrieval concepts in vector spaces. ACM Trans. Database Syst. 12 (2), 299–321.

Zipf, G., 1949. Human Behavior and the Principle of Least Effort. Addison-Wesley.