Cyber Defense Overview

Attack Patterns

John Franco
Electrical and Computer Engineering
Computer Science
Attack Patterns

What:
An abstraction mechanism for describing how a type of observed attack is executed.
Provides a description of the context where it is applicable and gives recommended methods of mitigating the attack.
A blueprint to control an exploit: minimize damage, preserve evidence, provide quick and efficient recovery, prevent similar future events, and gain insight into threats against the organization.

Why:
To be effective in preventing and dealing with attacks, the community needs to have a firm grasp of the attacker’s perspective and the approaches used to exploit software.
We want to build security into the systems we develop and not 'add' security after the system is developed.

https://www.us-cert.gov/bsi/articles/knowledge/attack-patterns/attack-pattern-usage
https://capec.mitre.org
Attack Patterns

Attributes:

Pattern Name and Classification:
A unique, descriptive identifier for the pattern.

Attack Prerequisites:
What conditions must exist or what functionality and what characteristics must the target software have, or what behavior must it exhibit, for this attack to succeed?

Description:
A description of the attack including the chain of actions taken.

Related Vulnerabilities or Weaknesses:
What specific vulnerabilities or weaknesses does this attack leverage?

Specific vulnerabilities reference industry-standard identifiers (CVE number, US-CERT number, etc.)

Underlying issues that may cause vulnerabilities (weaknesses) also reference industry-standard identifiers (CWE number)
Attack Patterns

Attributes:

Method of Attack:
What is the vector of attack used (e.g., malicious data entry, maliciously crafted file, protocol corruption)?

Attack Motivation-Consequences:
What is the attacker trying to achieve by using this attack? This is not the end goal of the attack within the target context but rather the specific technical result desired that could be leveraged to achieve the end mission objective. This information is useful for aligning attack patterns to threat models and for determining which attack patterns from the broader set available are relevant for a given context.

Attacker Skill or Knowledge Required:
What level of skill or specific knowledge must the attacker have to execute such an attack? Use rough scale (e.g., low, moderate, high) State contextual detail of what type of skills or knowledge are required.
Attack Patterns

Attributes:

Resources Required:
What resources (e.g., CPU cycles, IP addresses, tools, time) are required to execute the attack?

Solutions and Mitigations:
What actions or approaches are recommended to mitigate this attack, either through resistance or through resiliency?

Context Description:
In what technical contexts (e.g., platform, OS, language, architectural paradigm) is this pattern relevant?
This information is useful for selecting a set of attack patterns that are appropriate for a given context.

References:
What further sources of information are available to describe this attack?

Common Attack Pattern Enumeration & Classification: https://capec.mitre.org/
**Attack Patterns**

**Example 1:** Exploiting Software, G. Hoglund, G. McGraw, 2004

**Name and classification:**
Make the client invisible.

**Attack Prerequisites:**
The application must have a multi-tiered architecture with a division between client and server.

**Description:**
This attack pattern exploits client-side trust issues that are apparent in the software architecture. The attacker removes the client from the communication loop by communicating directly with the server. This could be done by bypassing the client or by creating a malicious impersonation of the client.

**References:**
- CWE-300 Man-in-the-Middle (MITM)
- CWE-346 Origin Validation Error
- CWE-290 Authentication Bypass by Spoofing
- CWE-306 Missing Authentication for Critical Function
- CWE-301 Reflection Attack in an Authentication Protocol
Example 1:

Method of Attack:
Direct protocol communication with the server

Attack Motivation and Consequences:
Potential information leak, data modification, arbitrary code execution, etc. These can all be achieved by bypassing authentication and filtering, accomplished with this attack pattern. The application must have a multi-tiered architecture with a division between client and server.

Resources Required:
None, although protocol analysis tools and client impersonation tools such as netcat can greatly increase the ease and effectiveness of the attack.
Attack Patterns

Example 1:

Solutions and Mitigations:

*Increase Resistance to Attack:*
Utilize strong two-way authentication for all communication between client and server. This option could have significant performance implications.

*Increase Resilience to Attack:*
ability to anticipate, withstand, recover from and adapt to adverse conditions, stresses, attacks, or compromises on systems that use or are enabled by cyber resources

Minimize the amount of logic and filtering present on client; place it instead on the server.
Use white lists on server to filter and validate client input.

Context Description:
Any raw data that exist outside the server software cannot and should not be trusted. Assume all clients will be hacked. The real problem is client-side trust. Accepting anything blindly from the client and trusting it is a bad idea, and yet this is often the case in server-side design

See xss.java example
Attack Patterns


Name and classification:
Shell Command Injection—Command Delimiters

Attack Prerequisites:
The application must pass user input directly into a shell Command.

See example_2.c and run-example-2
**Example 2:**

**Description:**
Using the semicolon or other off-nominal characters, multiple commands can be strung together. Unsuspecting target programs will execute all the commands. An example→ authenticating a user using a web form where the username is passed directly to the shell as in:

```exec("cat data_log_" + userInput + ".dat")```
The "+" sign denotes concatenation. The developer expects that the user will only provide a username. But, a malicious user could supply "username.dat; rm -rf /;" as the input to execute the malicious commands on the machine running the target software. Similar techniques are used in other attacks such as SQL injection. Command executed above is

```cat data_log_username.dat; rm -rf /; .dat```
The 'cat' may or may not succeed; the 'rm' will delete everything on the file system to which the application has access, and what happens with '.dat' is likely not relevant.
Attack Patterns

Example 2:

Related Vulnerabilities and Weaknesses:
- CWE-77/78  OS Command Injection
- CVE-1999-0043  Command execution via shell metachars
- CVE-1999-0067  phf CGI allows shell execution via metachars
- CVE-1999-0097  FTP client shell execution via metachars
- CVE-1999-0152  Finger daemon allows shell execution via metachars
- CVE-1999-0210  Privilege escalation via automountd & metachars
- CVE-1999-0260  jj CGI prog allows shell execution via metachars
- CVE-1999-0262  Linux CGI script → remote execution of anything
- CVE-1999-0279  EWS allows shell execution via metachars
- CVE-1999-0365  Metamail allows shell execution via metachars

Method of Attack:
By injecting other shell commands into other data that are passed directly into a shell command.

Attack Motivation-Consequences:
Execution of arbitrary code. Attacker uses target software with elevated privilege to execute otherwise protected commands.
Example 2:

**Attacker Skill or Knowledge Required:**
Finding and exploiting this vulnerability does not require much skill. A novice with some knowledge of shell commands and delimiters can perform a very destructive attack. A skilled attacker, however, may be required to subvert simple countermeasures such as rudimentary input filtering.

**Resources Required:**
No special or extensive resources are required for this attack.

**Solutions and Mitigations:**
Define valid inputs to all fields and ensure that the user input is always valid. Also perform white-list and/or black-list filtering as a backup to filter out known command delimiters.

**Context Description:**
OS: UNIX.

**References:**
Attack Patterns

Also For Fun:

Attack Pattern Glossary:
https://www.us-cert.gov/bsi/articles/knowledge/attack-patterns/attack-pattern-glossary
Attack Patterns

Attack Pattern Usage:
Attack Patterns can be leveraged for the production of safe code During all phases of the Software Development Life Cycle

Functional Requirements:
Include high-level requirements such as “users will be able to access the site using at least the latest versions of Chrome, Edge, and Mozilla Firefox” and “users shall be able to purchase books in any currency”.

Generally lead to more detailed functional requirements and can potentially drive out security requirements.

These security requirements can be functional or not functional in nature, but equally important.

Detailed functional and non-functional requirements including security requirements are often overlooked and neglected because the general focus is basic functionality.

Ex: next page
Attack Patterns

Attack Pattern Usage:

Derive Security Requirements from Functional Requirements:

Operational questions arise naturally from the functional specs:

1. If user views website with browser other than latest version of Chrome, Edge, or Firefox, what should happen?
2. Is it acceptable if the browser crashes?
3. Is it acceptable if absolutely nothing is displayed?
4. Is there anything that the server needs to do to differentiate between browsers?
5. What happens if the self-identification data sent by client is spoofed (e.g., if Firefox is set to report itself as Opera)?
6. If users can purchase books in other currencies, then should they be able to browse the website in other languages or encoding schemes?
7. If so, how many languages and encoding schemes should the website support?
8. What should happen if a client sends characters from a language or encoding scheme that the server doesn't accept?
Attack Patterns

Attack Pattern Usage:

Derive Security Requirements from Functional Requirements:
Process of making functional requirements more specific is an effective mechanism for identifying security requirements.

Consider question 8.

Person writing functional spec may respond with:
  If a client sends characters from a language that the server does not recognize, then the server will return a HTTP 415 status code which is now a security (well, safety) requirement.

This informs the developers how to handle the issue raised by question 8.

Otherwise, the problem might be overlooked, causing issues such as attackers being able to bypass input filters.
Attack Patterns

Attack Pattern Usage:
The Role of Attack Patterns in Deriving SR from FR:
The functional specifications contain keywords that may match developed attack patterns.

Example: customer says “the application must accept ASCII characters.” This triggers looking at the “Unicode Encoding” attack pattern

https://www.owasp.org/index.php/Unicode_Encoding

This raises the question:
What should the application do if Unicode characters or other unacceptable character set is encountered?

From this question, misuse/abuse cases can be defined such as malicious user provides Unicode chars to the data entry field

This gives designers a clear understanding of the environment they are designing for: they will be aware of this when designing

This also infers a security requirement: the system will filter all input for Unicode characters (could be overlooked otherwise)
Attack Patterns

Attack Pattern Usage:
The Role of Attack Patterns in Deriving SR from FR:
Many vulnerabilities are due to vague specs and requirements. This includes "unspecified behavior" in certain specifications (e.g., C language and how compilers must deal with certain situations or RFCs such as IP fragmentation and how end nodes interpret the specification in varying fashions).

Requirements should specifically address these ambiguities to avoid opening up multiple security holes.

Attack patterns allow the requirements gatherer to ask 'what if' questions to make the requirements more specific.

If an attack pattern states
   *Condition X can be leveraged by an attacker to cause Y*
then a valid question may be
   *What should the application do if it encounters condition X?*
Attack Patterns

Attack Pattern Usage:

Architecture and Design:
Decisions must be made regarding
how software is to be structured
how the various components will integrate and interact
which technologies will be leveraged
how requirements defining software function will be interpreted

Note: 50% of mistakes leading to security flaws are from design
Attack Patterns

Attack Pattern Usage:

Architecture and Design:

Consider 3-tier architecture:
- Client: web browser leveraging javascript and html
- WebServer: leveraging java® servlets
- Database server: leveraging Oracle 10i

Some Attack Patterns describe attacks exploiting arch flaws
Ex: Make the Client Invisible exploits client-side trust issues

According to this Attack Pattern:
- Nothing sent back by the client can be trusted even under SSL
- An attacker can spoof a client and send back anything
- All input validation, authorization checks, etc. must therefore be performed on the server side.
- Data that the client should not see should never be sent to it
- Performing authorization checks on the client side to determine what data to display is unacceptable.
Attack Patterns

Attack Pattern Usage:

Implementation and Coding:

In theory, each developer implementing the design should be writing well-defined components with well-defined interfaces at this point.

Attack Patterns allow developers to ensure that known weaknesses do not creep into the code they write.

But the relevant Attack Patterns must be found.

*Array Out of Bounds* attack pattern not relevant for Java.

Consider: leveraging an attack pattern such as *Simple Script Injection* to avoid XSS vulnerabilities.

Method: identify all places from which output is being sent to the user from an untrusted source and convert potentially dangerous characters into their HTML equivalents (e.g. `<` → `<` &lt;`).

Malicious data could include artifacts such as `<script>` tags inserted by an attacker (see example).
Attack Patterns

Software Testing and Quality Assurance:

**Goal:** attempt to break software so that the discovered issues can be fixed before an attacker can find them.

**Purpose of attack patterns:** have testers act as attackers

**Unit Testing:**
Test components independently to ensure they meet specs

Attack Patterns identify relevant targeted weaknesses and support test case generation for each component

**Example:** to test for shell command injection using command delimiters, malicious input strings containing delimiter separated shell commands can be crafted and input to the relevant components to ensure proper behavior.
Attack Patterns

Software Testing and Quality Assurance:

Integration Testing:
Test components together to ensure interfaces match and functionality does not conflict

Security issue: do different components make different assumptions regarding security that cause ambiguity or conflict?

Example: the Make the Client Invisible attack pattern can be used to create test cases that simulate an attacker bypassing the client and communicating directly with the server or an attacker impersonating the client to send malicious data to the server
**Attack Patterns**

**Software Testing and Quality Assurance:**

**System Testing:**
Test entire system to ensure system specs are satisfied

**Attack Patterns:** test the attack patterns used in requirements gathering phase.

**Example:** the *Unicode Encoding* attack pattern can be used to generate test cases that ensure that the system behaves properly when provided with unexpected characters.

Testers provide characters that the application is not supposed to accept to the application to see how it behaves. The application’s actual behavior when under attack should be compared with the desired behavior defined in the security requirements.
Attack Patterns

Software Testing and Quality Assurance:

Regression Testing:
Run tests when code is changed to make sure there is no (or unauthorized) change in behavior

**Attack Patterns:** same as above – nothing new here
Attack Patterns

Software Testing and Quality Assurance:

Testing in the Operational Environment:

For software to be useful it must be accessible – that means the bad guys have a chance at using it too, especially in an operational environment that is insecure (e.g. foreign spies have been employed at the NSA – they started out as good guys – U.S. citizens but they were recruited through money)

Software can be designed to be rock-solid secure but the operators, even well-intentioned, may cause disaster

http://www.bbc.co.uk/ahistoryoftheworld/objects/K30dyUL5RVupvcltH6qlWw
https://www.codesandciphers.org.uk/lorenz/fish.htm

Attack Patterns: penetration testing based on identified attack patterns – find implementation security failures

White box testing (everything is known about the system) against security requirements developed above – uncover unexpected architecture/design and implementation issues
Attack Patterns

Software Testing and Quality Assurance:
Testing in the Operational Environment:

#6 USS Yorktown dead in the water, 1998

https://www.slideshare.net/sommervi/ariane-5-launcher-failure?next_slideshow=1
https://medium.com/dataseries/when-smart-ships-divide-by-zero-uss-yorktown-4e53837f75b2
Attack Patterns

Software Testing and Quality Assurance:

Black Box Testing (penetration testing):

Black-box testing tools cannot determine the meaning of data (e.g. whether it is a social security number) and cannot apply business logic.

Attack patterns: those that can be executed remotely without requiring many steps.

Examples: cross-site scripting using injection of JavaScript in a HTTP parameter and SQL injection using separator characters.

Automated tools can be used to create tests, such as where a separator character is inserted into a HTML form field, to observe whether a database error occurs.
Attack Patterns

Software Testing and Quality Assurance:

White Box Testing:
Extensive analysis performed by security experts that have access to the software’s requirements, architecture, design, and code. Should get better results from white box testing. Testing may span months!

Attack patterns: determine areas of system risk hence areas of the system the white-box analysis should focus.

Attack patterns most effective: those that target architecture and design weaknesses.

Attack patterns useful for finding implementation weaknesses are used because weaknesses are found using code reviews

Example: Sniffing Sensitive Data on an Insecure Channel. Allows to determine if some information that should always be communicated over an encrypted channel is sent over an insecure channel. Since this issue is specific to a deployed environment, analysis of the deployed software is required
Attack Patterns

Software Testing and Quality Assurance:

System Operation:

**Attack patterns**: can guide design of secure operational configurations and procedures.

Operational knowledge of security issues observed in the deployed system can be used to feed back into the attack pattern generation process.

Improved operational procedures for security in the field may be found using attack patterns that emulate what an attacker might try.

In case of a successful exploit, investigation may result in new attack patterns that can be used to change configurations or operating procedures and be applied to redesign to prevent this type of attack being successful in the future.
Attack Patterns

Software Testing and Quality Assurance:

System Operation:

**Needed Because**: a large part of environmental conditions consist of operational configurations and procedures.

Vulnerable code exists in the field:
- maybe the vulnerabilities are known but cost too much to fix
- may cost less to mitigate attacks in the field instead of preventing problems in design