Welcome

Python – TDD & Cybersecurity

Hello

About me...

- 25+ years in the indusry
- 20+ years in teaching
- Certified Cloud architect
- Passionate about learning
- Also, passionate about Reese's Cups!

Prerequisites

This course assumes you:

- Are familiar with the Python language (v3+)
- Are looking to build knowledge in Test Driven Development (TDD) & security concepts

Why study this subject?

- Testing (especially unit testing) is critical to software quality
- Designing for testability is an “art”
- In the modern digital age, we seek to “shift security left”

We teach over 400 technology topics

You experience our impact on a daily basis!

My pledge to you

I will...

- Make this interactive
- Ask you questions
- Ensure everyone can speak
- Use an on-screen timer

Objectives

At the end of this course you will be able to:

- Describe some common security attacks and threat modeling strategies
- Define Test Driven Development (TDD) and talk about its implementation
- Define DevSecOps and talk about theoretical and practical aspects of implementing

Agenda

- Cybersecurity & Threat Modeling – 1 day
- Test Driven Development (TDD) – 1.5 days
- Python Testing & DevSecOps – 1.5 days

How we’re going to work together

- Slides / lecture
- Demos
- Team discussions
- Labs

Introduction

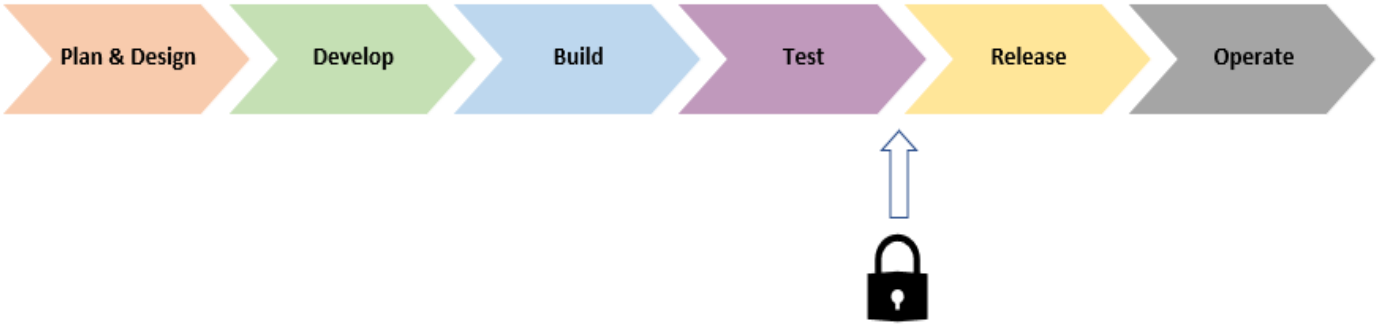
How Software Engineers Test (Sometimes)

- Manually
- In a semi-automated fashion, but late in the SDLC
- In large blocks of functionality requiring complex coordination

So, Why Is That a Problem?

- Issues found late in the development workflow are more expensive to fix
- Tests requiring complex coordination and setup are brittle
- Without automation, testing takes more time and is more costly
- Testing can be hard to “retrofit” into a design not built for it

How Software Engineers Think About Security (Sometimes)



How Software Engineers Think About Security (Sometimes)

- An afterthought
- As a concern right before release
- “Not my area of expertise…”

So, Why Is That a Problem?

- Issues found late in the development workflow are more expensive to fix
- Security issues can be complex to remediate and endanger release schedules
- Potential for significant financial and/or reputational damage

Opportunities

- Raising awareness of common threats & attack vectors
- Practicing threat modeling techniques to help assess and focus
- Learning to design for testability
- Integrating quality gates for testing and security into your CI/CD pipelines

Cybersecurity & Threat Modeling

Common Security Attacks

Common Security Attacks

- Denial of Service (DoS or DDos)
- SQL injection
- Large files
- Cross Site Scripting (XSS)
- Credential stuffing

Denial of Service (DoS)

- Server “flooded” with so much bogus traffic that systems are unable to serve valid requests
- Alternatively, instruction(s) received that trigger a server or system “crash”

Denial of Service (DoS)

Common flooding attacks:

- Buffer overflow (most common type)
- ICMP flood (AKA “smurf attack” or “ping of death”)
- SYN flood

Denial of Service (DoS)

Crash attacks:

- Often involves send of data targeting common classes of bug
- Request used to crash or severely destabilize the system

Distributed Denial of Service (DDoS)

- Similar profile as DoS but uses multiple systems to orchestrate the attack
- Provides attacker with advantages

Distributed Denial of Service (DDoS)

Potential advantages for attacker:

- More agents means more power behind the attack
- Location of attack is difficult to detect (often globally-routed)
- Easier to shut down a single attack machine than multiple
- Identity of attacker is more easily disguised

Defending Against Denial of Service

- Ensure service has good AuthN/AuthZ in place
- Utilize a proxy or gateway and configure throttling
- In Production, disable ICMP pings or use rate-limit for ICMP requests (e.g., using iptables)

Defending Against Denial of Service

- Minimize server resources allocated to an incoming sync request
- Use a form of “cookie” for managing sync requests
- SYN cookies and RST cookies are examples
- Mitigation often involves configuration of network components or layers

Denial of Service - Demo

SQL Injection

- Attacker injects SQL (Structured Query Language) queries into app flow (e.g., UI)
- In some cases, injected SQL used to retrieve additional sensitive detail
- In other cases, injected SQL used to alter or damage a company’s critical data

Types of SQL Injection Attack

- In-band
- Blind

In-band SQL Injection

- Uses existing channel of communication for an attack
- Error-based – attacker performs actions that cause errors in order to gather “intel” about database structure
- Union-based – attacker takes advantage of UNION SQL operator to fuse multiple SELECT statements into single response

Blind SQL Injection

- Attacker sends separate, independent data queries to a server
- Called blind because results of query are not sent back to attacker
- Instead, attacker observes results to infer vulnerabilities

Blind SQL Injection

- Relies on response and behavior patterns of server so slower to execute
- Boolean – attacker sends SQL query to database and determines if attack valid based on response received (true or false)
- Time-based – attacker sends SQL query to database and determines if attack valid based on amount of time taken to process

Defending Against SQL Injection

- Use well-defined contracts to explicitly map expected results from application
- Sanitize inputs and use parameterized queries in code
- Use a Web Application Firewall that includes protections at the application layer (including protection against SQL Injection)

SQL Injection - Lab

Large Files

- Sometimes resembles another form of Denial of Service
- Attacker attempts to send one (or several) very large files as upload
- Could also occur with extremely large payloads (JSON or XML bodies)
- As a result, network connectivity to servers or services can become “clogged”

Defending Against Large Files

- Use configuration to limit file/payload sizes and number of concurrent connections from a client
- Utilize timeouts judiciously to prevent large file operations from completing
- Can also leverage MIME types as a way to limit acceptable types of data
- Finally, proxies or gateways (WAF) can be configured for mitigation at the network layer

Cross Site Scripting (XSS)

- A type of injection – but script instead of SQL
- Attacker uses inputs to attempt injection of a <script>…</script> element
- An example could be posting a comment with a link that routes to a malicious site

Cross Site Scripting (XSS)

- Without inspection for malicious content, <script> can be returned (and executed) in user's browser
- Malicious content can include JavaScript, HTML, Flash, etc.
- Really, any code that browser can execute

Cross Site Scripting (XSS)

Common forms of attack:

- Stealing cookie or session information
- Redirects to web content controlled by an attacker
- Executing malicious operations on user’s machine
- Leveraging impersonation for elevated privilege

Cross Site Scripting (XSS)

Stored XSS attacks:

- Injected script permanently stored on target servers
- Could include storage in database, forum, comment field, etc.
- Malicious script returned to browser as part of retrieval from storage

Cross Site Scripting (XSS)

Reflected XSS attacks:

- Malicious script is indirectly transferred back to browser
- When user clicks on malicious link, code gets injected into vulnerable site
- Malicious code then reflected back to user under the cover of “valid” site interaction for immediate execution

Cross Site Scripting (XSS)

DOM-based XSS attacks:

- Takes advantage of sites that copy input to DOM without validation
- Similar to reflected in that victim is tricked into sending malicious code to vulnerable site
- However, input lands in DOM in the browser for execution instead of being reflected back

Defending Against Cross Site Scripting

- Encode and validate everywhere – do not trust user inputs, escape outputs, and manage response headers
- Use libraries with utility handlers where possible (e.g., Jinja or Django)
- Quote every attribute of every tag in HTML

Defending Against Cross Site Scripting

- Use HttpOnly directive on custom cookie response headers (i.e., “Set-Cookie” header)
- Use the “X-Content-Type-Options: nosniff” to prevent MIME type sniffing (i.e., dynamic changes to Content-Type header)
- Leverage network components like WAF with built-in protection to intersect at the network layer

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Cross Site Scripting - Demo

Credential Stuffing

- Attackers use lists of compromised credentials to try and find a breach
- Based on assumption that many users reuse same credentials across sites
- Uses bots, automation, and scale

Credential Stuffing

- Like a brute force attack
- However, instead of random strings, uses existing lists of known credentials
- Powered by broad availability of compromised info and increasing sophistication of bots & automation

Credential Stuffing

- Attacker sets up bot able to attempt login for multiple accounts in parallel
- Uses an automated process to test effectiveness
- Monitors for breaches and pulls/retains sensitive detail when found
- With parallel attempts, often fakes IP addresses to make difficult to trace

Defending Against Credential Stuffing

- Leverage MFA (Multi-Factor Authentication)
- Use a CAPTCHA (though I hate them!)
- Gather details about user devices to create a “fingerprint” for incoming sessions – if same “fingerprint” is logged several times in sequence, block

Defending Against Credential Stuffing

- Leverage IP blacklisting
- Rate-limit non-residential traffic sources (like public Cloud)
- Block headless browsers based on JavaScript calls used
- Disallow e-mail addresses as user IDs

OWASP Top 10

- See https://owasp.org/Top10/
- Can change from year-to-year so be aware

Threat Modeling

Standards & Compliance

Can include:

- By geographical location
- By industry
- By technology

Standards & Compliance by Region

- Standards and compliance enforcement can vary by area of the world
- For example, EU likely has different requirements than US (e.g., General Data Protection Regulation or GDPR in EU)

Standards & Compliance by Region

Can include considerations for:

- How data is transmitted
- How data is secured, managed, & used
- Physical or systems security

Standards & Compliance by Region

- Failure to adhere can limit ability to do business in the region
- Or can result in significant penalties and/or reputational damage
- Can add permutations to approach to build out of the tech

Standards & Compliance by Industry

- Different industries may have different regulations
- There can also be a difference in physical requirements
- Think remote oil field vs. data center vs. nuclear power plant

Standards & Compliance by Industry

- Regulations often driven by types of data being gathered
- Medical devices likely subject to HIPAA regulations
- Point-of-Sale (POS) devices likely require PCI compliance

Standards & Compliance by Industry

- Depending on the industry, failure to comply may have devastating impact
- Think potential exposure for communications backbone, air traffic control, or autonomous vehicles

Standards & Compliance by Technology

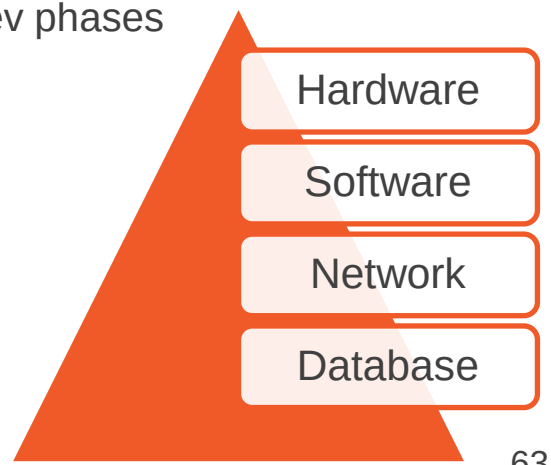
- Each technology stack brings with it different layers of exposure
- Requires vigilance, monitoring, & willingness to adjust as and when needed
- Areas of potential exposure range from OS to runtime to Open-Source libraries used in application

Assessing Security Risks

- To secure a solution, attack surfaces and potential threats must be identified
- Common practice utilizes something called threat modeling
- Includes modeling and analyzing possible attack vectors based on application

Assessing Security Risks

- Risk assessment should account for different “zones” of execution
- Security requirements must be understood in context of specific use cases
- Ideally, threat modeling would be executed during design & dev phases



Threat Modeling

Threat Modeling

Can be viewed in two different, but related, contexts:

- Implementation of controls mapped to security requirements & policy (prevention)
- Implementation of countermeasures against possible known attacks (remediation)

Threat Modeling

Multiple approaches:

- Attacker-centric (think like an attacker!)
- Asset-centric (what do we have to lose?)
- Application-centric (what are we building & testing?)

Threat Modeling – Attacker-Centric

- Involves profiling potential attacker’s characteristics, skillset, & motivation
- Grouped by type of attacker and, hence, type of attacks
- Examples include attacker looking to steal sensitive information, hold a company’s data or systems “hostage”, or disrupt service

Threat Modeling – Attacker-Centric

Utilizes tree diagrams to map combinations:

- Goals of attacker
- Specific system-related considerations
- Potential attack methods
- Means for detection/mitigation

Threat Modeling – Attacker-Centric

- In some cases, potential attacks can be mapped to known patterns
- Goal is to view your system (and its vulnerabilities) through the eyes of a “bad actor” looking to attack

Threat Modeling – Asset-Centric

- Involves identifying the specific software & data assets of an organization
- Data assets often classified by sensitivity and intrinsic value to an attacker – helps with prioritization
- Uses attack graphs to visually illustrate patterns of potential attack against a given asset

Threat Modeling – Application-Centric

- Involves the security design of the system
- Security requirements (like other application requirements) gathered and prioritized
- Application functionality built to address the requirements (SDLC)
- Frameworks exist to assist a team with asking the right questions

Threat Modeling – Key Considerations

Valuable principles:

- Defense in Depth
- Principle of Least Privilege
- Secure by Default

Securing Data in Motion

- Security required as data flows through the ether between producer and consumer

If attacker able to intercept information flowing between the two:

- Potentially exposes sensitive information contained within header or payload
- Could allow insertion of alternate, damaging alternative information

Securing Data in Motion

- Certificate/secrets-based Transport Layer Security (TLS) can be used to protect
- Leverages certs/keys to encrypt data “on the wire”

Securing Data at Rest

If data at rest (stored & aggregated data) compromised:

- May lead to inaccurate conclusions from analysis
- Could provide competitor or bad actor access to a company’s competitive advantage

As with “in motion”, certificate-based encryption in storage is key

Threat Modeling Tools & Frameworks

OWASP Threat Dragon

https://owasp.org/www-project-threat-dragon/

PASTA

https://www.cynance.co/pasta-threat-modelling/

OCTAVE

https://www.pluralsight.com/guides/cybersecurity-threat-modeling-with-octave

STRIDE

https://securityintelligence.com/articles/what-is-stride-threat-modeling-anticipate-cyberattacks/

Comparison of Various Options

https://www.eccouncil.org/threat-modeling/

https://www.techwell.com/techwell-insights/2020/05/choosing-right-threat-modeling-methodology

Thank you!

If you have additional questions, please reach out to me at: (asanders@gamuttechnologysvcs.com)

DevelopIntelligence
A PLURALSIGHT COMPANY