





Welcome to

# 10. Defensive: Defensive Injection and DoS

Security in Web Development Elective, KEA

Henrik Kramselund he/him han/ham hkj@zencurity.com @kramse  

Slides are available as PDF, kramse@Github

10-defensive-injection-and-dos-security-in-web.tex in the repo security-courses

# Goals for today



Today's goals:

- Doing defensive for protecting the backend system
- Defending Against Injection and DoS Attacks

Photo by Thomas Galler on Unsplash

# Plan for today



Subjects, defending against:

- SQL injection protection
- Denial of Service (DoS)
- Distributed Denial of Service (DDoS)

Exercises

- Django ORM
- SSL/TLS overloading with THC tool
- Monitoring with Packetbeat, reading exercise
- Django related exercises
- Code searching and viewing

## Time schedule



- 1) SQL injection protection 45min
- 2) DoS/DDoS 45min
- 3) Program Building blocks 45min
- 4) Actual code – JuiceShop and Django 45min

As always we will not follow this to the minute, but be flexible

# Reading Summary



*Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

Part III. Defense, chapters 19-21

25. Defending Against Injection

26. Defending Against DoS

## 25. Defending Against Injection



Previously we discussed the risk that injection-style attacks bring against web applications. These attacks are still common (although more common in the past), typically as a result of improper attention on the part of the developer writing any type of automation involving a CLI and user-submitted data.

Injection attacks also cover a wide surface area. Injection can be used against CLIs or any other isolated interpreter running on the server (when it hits the OS level, it becomes command injection instead). As a result, when considering how we will defend against injection-style attacks, it is easier to break such defensive measures up into a few categories.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

## PHP shell escapes



PHP in the old days was horrible insecure. It was possible to include files from HTTP urls, file URL open etc. Often user input was copied directly into a shell, which is why we also spend some time talking about shells and Linux/Unix

```
<pre>
<?php passthru(" netstat -an && ifconfig -a"); ?>
</pre>
```

Other tools have similar shell escapes:

- Perl: `print `/usr/bin/finger $input{'command'}`;`
- UNIX shell: ``echo hello``
- Microsoft SQL: `exec master..xp_cmdshell 'net user test testpass /ADD'`

**Result: web server send back data through HTTP/HTTPS**

# Attacking Injection Summary – Which database



SQL databases are still the most popular general-purpose database on the market. SQL query language is strict, but reliably fast and easy to learn. SQL can be used for anything from storage of user credentials to managing JSON objects or small image blobs. The largest of these are PostgreSQL, Microsoft SQL Server, MySQL, and SQLite.

Source: *Web Application Security* page 49

- Identify the framework, operating system, dependencies
- Any error messages help an attacker
- Different methods for attacking MySQL versus Microsoft SQL Server



# Attacking Injection Summary – Identifying Databases



If database error messages are sent to the client directly, a similar technique to the one for detecting server packages can be used to determine the database. Often this is not the case, so you must find an alternative discovery route. ... One technique that can be used is primary key scanning. Most databases support the notion of a “primary key,” which refers to a key in a table (SQL) or document (NoSQL) that is generated automatically upon object creation and used for rapidly performing lookups in the database.

Source: *Web Application Security* page 95

- Then book goes on to describe MongoDB `_id` hexadecimal-compatible field of length 12

# Attacking Injection Summary – SQL Injection and PHP



One of the most commonly known types of attacks against a web application is SQL injection. SQL injection is a type of injection attack that specifically targets SQL databases, allowing a malicious user to either provide their own parameters to an existing SQL query, or to escape an SQL query and provide their own query. **Naturally, this typically results in a compromised database because of the escalated permissions the SQL interpreter is given by default. ...**

Old-school PHP developers would interweave a combination of SQL, HTML, and PHP into their PHP files—an organizational model supported by PHP that would be misused, resulting in an enormous amount of vulnerable PHP code. ...

In more recent years, PHP coding standards have become much more strict and the language has implemented tools to reduce the odds of SQL injection occurring.

Source: *Web Application Security* page 147-148

# Attacking Injection Summary – SQL Injection and Other languages



The result of these developments is that there is less SQL injection across the entire web. In fact, injection vulnerabilities have decreased from nearly 5% of all vulnerabilities in 2010 to less than 1% of all vulnerabilities found today, according to the National Vulnerability Database.

The security lessons learned from PHP have lived on in other languages, and it is much more difficult to find SQL injection vulnerabilities in today's web applications.

It is still possible, however, and still common in applications that do not make use of secure coding best practices.

Source: *Web Application Security* page 149

## Defend with Defense in Depth



**Definition 14-1** The *principle of least privilege* states that a subject should be given only those privileges that it needs in order to complete the task.

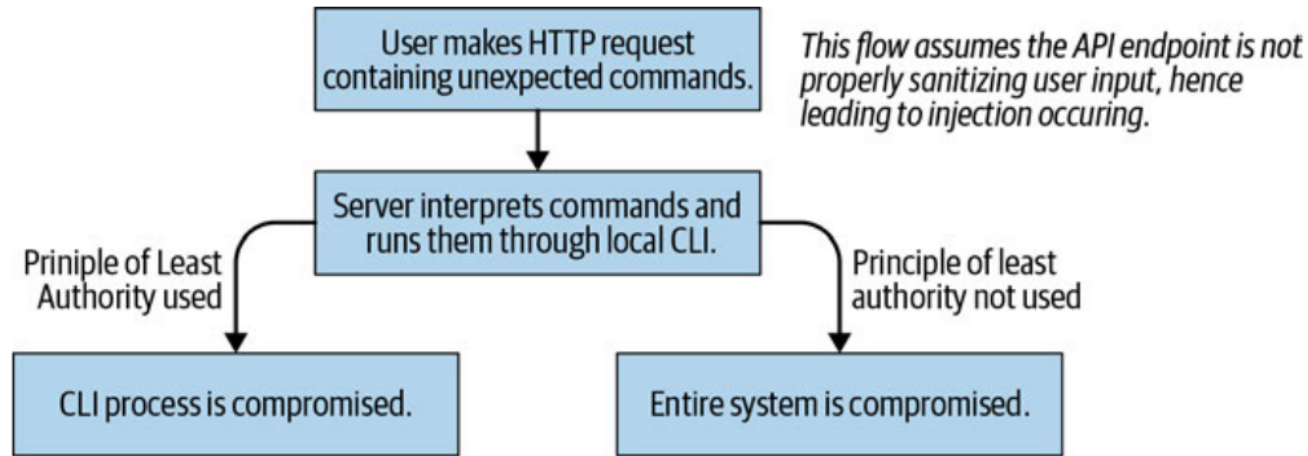
Also drop privileges when not needed anymore, relinquish rights immediately

Example, need to read a document - but not write.

Database systems can often provide very fine grained access to data

Quote from Matt Bishop, Computer Security

# Defend with Least Privilege



*Figure 25-1. Using the principle of least authority when designing your web application, you can reduce the impact of any injection attack that is accidentally introduced*

Source: *Web Application Security* page 264

# Role-based Access Control (RBAC)



In computer systems security, **role-based access control (RBAC)**[1][2] or role-based security[3] is an approach to restricting system access to unauthorized users. It is used by the majority of enterprises with more than 500 employees,[4] and can implement mandatory access control (MAC) or discretionary access control (DAC).

Role-based access control (RBAC) is a policy-neutral access-control mechanism defined around **roles and privileges**. The components of RBAC such as role-permissions, user-role and role-role relationships make it simple to perform user assignments. A study by NIST has demonstrated that RBAC addresses many needs of commercial and government organizations[citation needed]. RBAC can be used to facilitate administration of security in large organizations with hundreds of users and thousands of permissions. Although RBAC is different from MAC and DAC access control frameworks, it can enforce these policies without any complication.

Quote from [https://en.wikipedia.org/wiki/Role-based\\_access\\_control](https://en.wikipedia.org/wiki/Role-based_access_control)

# Example role based system: PostgreSQL security



	11	10	9.6	9.5	9.4
Channel binding for SCRAM authentication	Yes	No	No	No	No
Column level permissions	Yes	Yes	Yes	Yes	Yes
Default permissions	Yes	Yes	Yes	Yes	Yes
GRANT/REVOKE ON ALL TABLES/SEQUENCES/FUNCTIONS	Yes	Yes	Yes	Yes	Yes
GSSAPI support	Yes	Yes	Yes	Yes	Yes
Large object access controls	Yes	Yes	Yes	Yes	Yes
Native LDAP authentication	Yes	Yes	Yes	Yes	Yes
Native RADIUS authentication	Yes	Yes	Yes	Yes	Yes
Per user/database connection limits	Yes	Yes	Yes	Yes	Yes
ROLES	Yes	Yes	Yes	Yes	Yes
Row-Level Security	Yes	Yes	Yes	Yes	No
SCRAM-SHA-256 Authentication	Yes	Yes	No	No	No
Search+bind mode operation for LDAP authentication	Yes	Yes	Yes	Yes	Yes
security_barrier option on views	Yes	Yes	Yes	Yes	Yes
Security Service Provider Interface (SSPI)	Yes	Yes	Yes	Yes	Yes
SSL certificate validation in libpq	Yes	Yes	Yes	Yes	Yes
SSL client certificate authentication	Yes	Yes	Yes	Yes	Yes
SSPI authentication via GSSAPI	Yes	Yes	Yes	Yes	Yes

Feature overview security features in PostgreSQL

<https://www.postgresql.org/about/featurematrix/#security>

# Security by Design



- Doing the above should result in applications which are secure by design
- Adhering to the best security principles
- Implementing security from design to deployment ensure good security



## Prepared Statements – what it is



One development that most SQL implementations have begun to support is prepared statements. Prepared statements reduce a significant amount of risk when using user-supplied data in an SQL query. Beyond this, prepared statements are very easy to learn and make debugging SQL queries much easier.

Prepared statements work by compiling the query first, with placeholder values for variables. These are known as bind variables, but are often just referred to as placeholder variables. After compiling the query, the placeholders are replaced with the values provided by the developer. As a result of this two-step process, the intention of the query is set before any user-submitted data is considered.

Source: *Web Application Security* page 261

- Doing the above should result in applications which are secure by design
- Adhering to the best security principles
- Implementing security from design to deployment ensure good security

# Prepared Statements – what it provides and how to use



In MySQL, prepared statements are quite simple:

```
PREPARE q FROM 'SELECT name, barCode from products WHERE price <= ?';  
SET @price = 12;  
EXECUTE q USING @price;  
DEALLOCATE PREPARE q;
```

With a prepared statement, because the intention is set in stone prior to the user-submitted data being presented to the SQL interpreter, the query itself cannot change. **This means that a SELECT operation against users cannot be escaped and modified into a DELETE operation by any means.** An additional query cannot occur after the **SELECT** operation if the user escapes the original query and begins a new one. **Prepared statements eliminate most SQL injection risk** and are supported by almost every major SQL database: MySQL, Oracle, PostgreSQL, Microsoft SQL Server, etc.

Source: *Web Application Security* page 262

# Use Object–relational mapping (ORM)



Object–relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object–relational mapping, although some programmers opt to construct their own ORM tools.

Source: [https://en.wikipedia.org/wiki/Object%E2%80%93relational\\_mapping](https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping)

- Many complexities and problems can be avoided by the use of ORM
- I often recommend against writing SQL by hand nowadays, especially when prototyping

# Example Django ORM



```
from django.db import models
```

```
class Person(models.Model):  
    first_name = models.CharField(max_length=30)  
    last_name = models.CharField(max_length=30)
```

```
CREATE TABLE myapp_person (  
    "id" serial NOT NULL PRIMARY KEY,  
    "first_name" varchar(30) NOT NULL,  
    "last_name" varchar(30) NOT NULL  
);
```

Source: <https://docs.djangoproject.com/en/4.0/topics/db/models/>

- First create a data model, then let Django do the hard part
- Search for your favourite language

# Django SQL injection protection



SQL injection protection¶ SQL injection is a type of attack where a malicious user is able to execute arbitrary SQL code on a database. This can result in records being deleted or data leakage.

Django's querysets are protected from SQL injection since their queries are constructed using query parameterization. A query's SQL code is defined separately from the query's parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver.

Django also gives developers power to write raw queries or execute custom sql. These capabilities should be used sparingly and you should always be careful to properly escape any parameters that the user can control. In addition, you should exercise caution when using `extra()` and `RawSQL`.

Source: <https://docs.djangoproject.com/en/4.0/topics/security/>

- Django was released in July 2005 – so many years of maturing

# Exercise



Now lets do the exercise

## Django ORM 20 min

which is number **44** in the exercise PDF.

## 26. Defending Against DoS



DoS attacks usually involve the **use of system resources**, which can make detecting them a bit difficult without robust server logging. It can be **difficult to detect a DoS attack that occurred in the past** if it came through legitimate channels (such as an API endpoint). As such, a first measure against DoS-style attacks should be **building up a comprehensive enough logging system** in your server that all requests are logged alongside their time to respond. You should also manually log the performance of any type of async “job-style functions, such as a backup that is called through your API but runs in the background and does not generate a response once it completes. Doing this will allow you to find any attempts (accidental or malicious) at exploiting a DoS vulnerability (server side) that would have otherwise been difficult and time-consuming.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

- Denial of service can be the result of any resource running out, CPU, memory, bandwidth, interrupt handling, session table, open files, open TCP connections, database connection, running out of random bits for keys, running out of file nodes in the file system, ...

# Protecting Against Regex DoS



Regex DoS attacks are likely the easiest form of DoS to defend against, but require prior knowledge of how the attacks are structured (as shown in Part II of this book). With a proper code review process, you can prevent regex DoS sinks (evil or malicious regex) from ever entering your codebase. ...

The second step is to make sure there are no places in your application where a **user-supplied regex is utilized**. Allowing user-uploaded regular expressions is like walking through a minefield and hoping you memorized the safe-route map correctly. It will take a huge coordinated effort to maintain such a system, and it is generally an all-around bad idea from a security perspective.

- Have a timeout for all potentially long running tasks, shut them down
- Measure time taken for requests, mark those taking too long – prevent them from running again, too often
- Any search form can be a trigger!



# Protecting Against Logical DoS



Logical DoS is much more difficult to detect and prevent than regex DoS. Much like regex DoS, logical DoS is not exploitable under most circumstances unless your developers accidentally introduce a segment of logic that can be abused to eat up system resources. ...

In order to protect against logical DoS, we need to identify the areas of our codebase in which critical system resources are utilized.

- I think client attacks in this category should include mining various cryptocurrencies

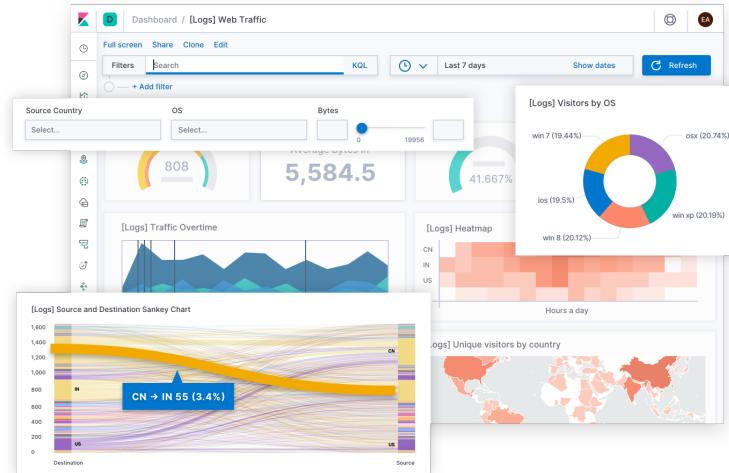
# Protecting Against DDoS



Distributed denial of service attacks (DDoS) are much more difficult to defend against than DoS attacks that originate from a single attacker. While single-target DoS attacks often target a bug in application code (like an improperly written regex, or a resource-hogging API call), DDoS attacks are usually much simpler by nature. Most DDoS attacks on the web originate from multiple sources, but are controlled by a centralized source. This is orchestrated via a single attacker or group of attackers who distribute malware by some channel.

- I have multiple presentations about this area, in general, it is hard to protect your site
- Many sites use cloudflare and similar services

# Elastic stack Kibana



Screenshot from <https://www.elastic.co/kibana>

Elasticsearch is a search engine and document store used in a lot of different systems, allowing cross application integration.

# Getting started with Elastic Stack



Easy to get started using the tutorial *Getting started with the Elastic Stack* :

<https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-elastic-stack.html>

Today Elastic Stack contains lots of different parts.

- Elasticsearch - the core engine
- Logstash - a tool for parsing logs and other data.

<https://www.elastic.co/logstash>

"Logstash dynamically ingests, transforms, and ships your data regardless of format or complexity. Derive structure from unstructured data with grok, decipher geo coordinates from IP addresses, anonymize or exclude sensitive fields, and ease overall processing."

- Kibana - a web application for accessing and working with data in Elasticsearch

<https://www.elastic.co/kibana>

# Designing and Standardizing HTTP Response Codes



- 100-199 are informational codes used as low-level signaling mechanisms, such as a confirmation of a request to change protocols
  - 200-299 are general success codes used to describe various kinds of success conditions
  - 300-399 are redirection codes used to request that the consumer retry a request to a different resource identifier, or via a different intermediary
  - 400-499 represent consumer-side error codes that indicate that the consumer has produced a request that is invalid for some reason, example 404 file not found
  - 500-599 represent service-side error codes that indicate that the consumer's request may have been valid but that the service has been unable to process it
- Elasticsearch exposes REST APIs that are used by the UI components and can be called directly to configure and access Elasticsearch features.

Source:

*Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

# Exercise



Now lets do the exercise

## Packetbeat monitoring 15 min

which is number **45** in the exercise PDF.

# Common DDoS attack types



A list of the most common DDoS attack types are:

- TCP SYN flooding
- TCP other flooding
- UDP flooding NTP, etc.
- ICMP flooding
- Misc - stranger attacks and illegal combinations of flags etc.

These can affect your services, directly on the servers or perhaps DNS, domains or other dependencies.

Multiple tools for producing packets exist, hping3 and some configurations for protecting - PF rules, switch rules, server firewall rules.

# hping3 packet generator



```
usage: hping3 host [options]
  -i --interval wait (uX for X microseconds, for example -i u1000)
  --fast      alias for -i u10000 (10 packets for second)
  --faster    alias for -i u1000 (100 packets for second)
  --flood     sent packets as fast as possible. Don't show replies.
...
hping3 is fully scriptable using the TCL language, and packets
can be received and sent via a binary or string representation
describing the packets.
```

- Hping3 packet generator is a very flexible tool to produce simulated DDoS traffic with specific characteristics
- Home page: <http://www.hping.org/hping3.html>
- Source repository <https://github.com/antirez/hping>

My primary DDoS testing tool, easy to get specific rate pps



# Various DoS tools

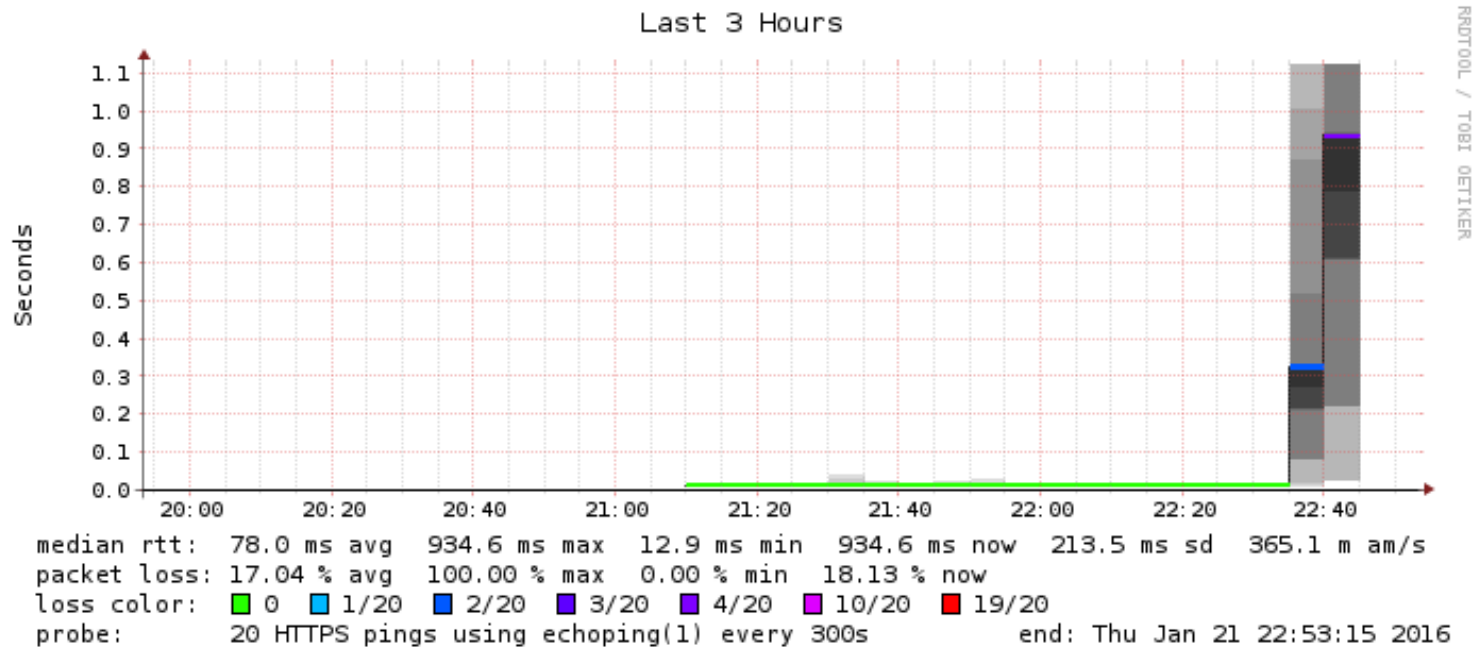


Small list of DoS/DDoS tools available on Kali Linux

GoldenEye Slowloris LOIC (Low Orbit Ion Cannon) HOIC (High Orbit Ion Cannon) THC-SSL-DoS HULK (http Unbearable Load King) Pyloris TOR's Hammer XOIC RUDY (R U Dead Yet ?) DAVOSET OWASP HTTP POST

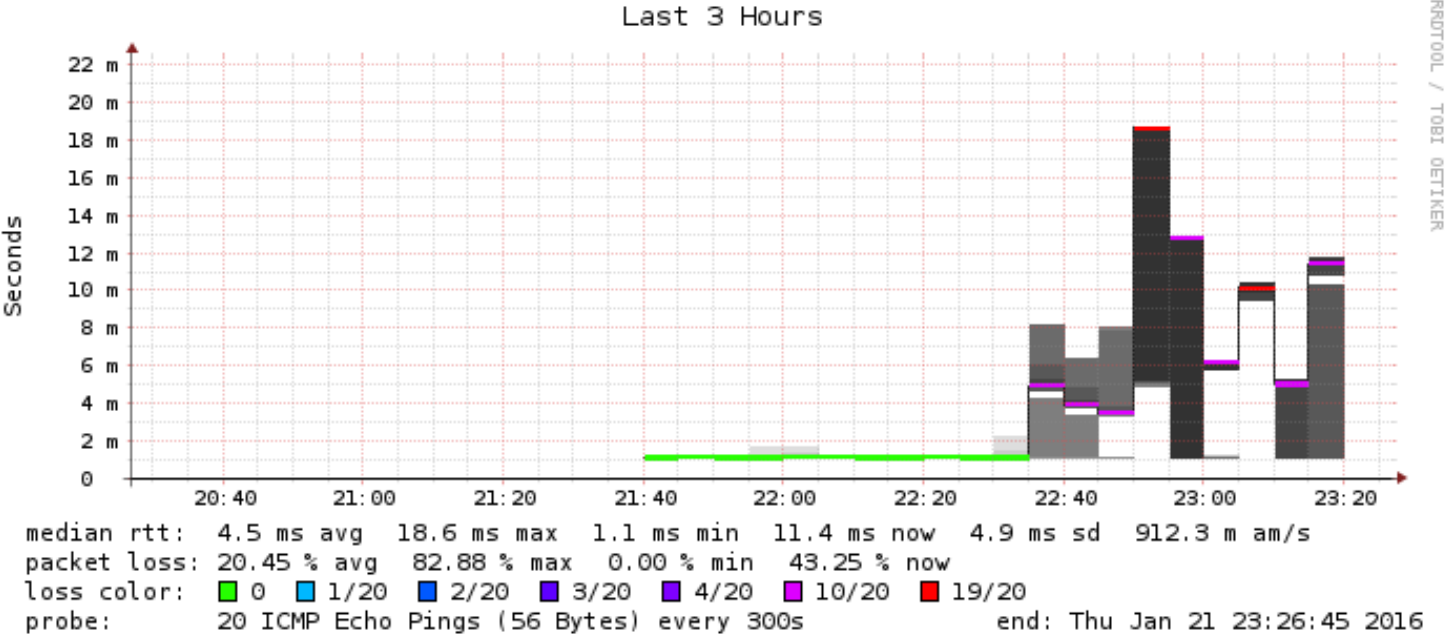
- Slowloris – keep sessions open, refresh just before timeout  
[https://en.wikipedia.org/wiki/Slowloris\\_\(computer\\_security\)](https://en.wikipedia.org/wiki/Slowloris_(computer_security))
- THC-SSL-DoS *Establishing a secure SSL connection requires 15x more processing power on the server than on the client. THC-SSL-DOS exploits this asymmetric property by overloading the server and knocking it off the Internet.*

# Rocky Horror Picture Show - 1



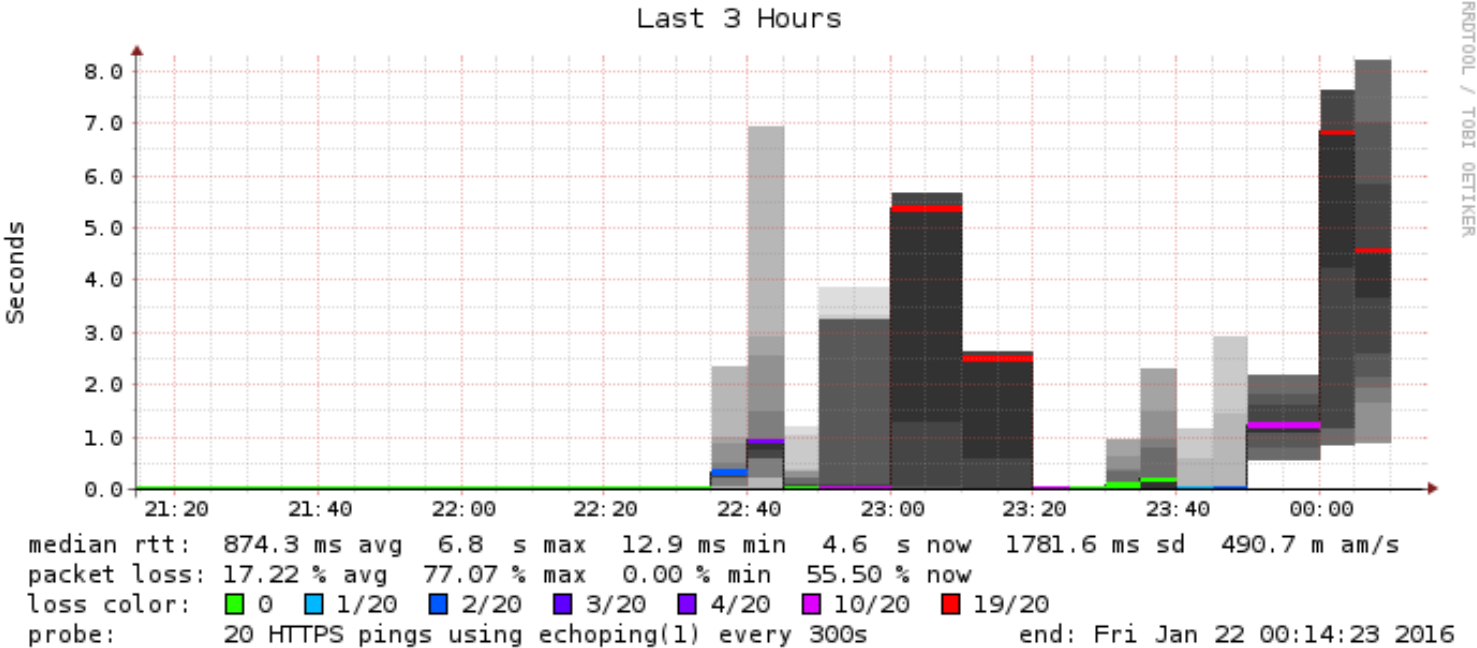
Why test? Results from small 50.000 pps SYN attack?

# Rocky Horror Picture Show - 2



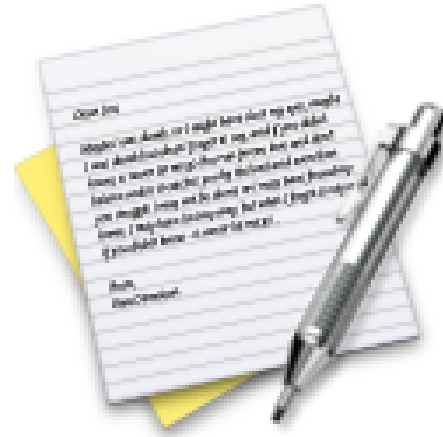
Oh no 500.000 pps UDP attacks work?

# Rocky Horror Picture Show - 3



Oh no spoofing attacks work?

# Exercise



Now lets do the exercise

## Bonus: TCP SYN flooding 30min

which is number **31** in the exercise PDF.

# Exercise



Now lets do the exercise

## THC SSL DoS 20 min

which is number **46** in the exercise PDF.

## Remember: Overview of Software Security



- Security Testing Versus Traditional Software Testing
- Functional testing does not prevent security issues!
- SQL Injection example, injecting commands into database
- Attackers try to break the application, server, operating system, etc.
- Use methods like user input, memory corruption / buffer overflow, poor exception handling, broken authentication, ...

Where to start?



The OWASP Top Ten provides a minimum standard for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are.

The Open Web Application Security Project (OWASP)

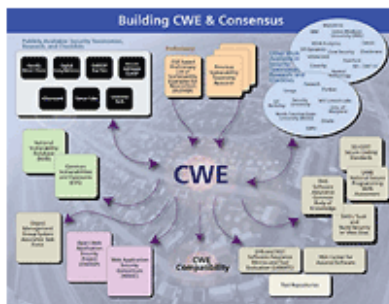
OWASP produces lists of the most common types of errors in web applications

<http://www.owasp.org>

Create Secure Software Development Lifecycle



# CWE Common Weakness Enumeration



[Enlarge](#)

**CWE™** International in scope and free for public use, CWE provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design.

## CWE in the Enterprise

- ▲ [Software Assurance](#)
- ▲ [Application Security](#)
- ▲ [Supply Chain Risk Management](#)
- ▲ [System Assessment](#)
- ▲ [Training](#)
- ▲ [Code Analysis](#)
- ▲ [Remediation & Mitigation](#)
- ▲ [NVD \(National Vulnerability Database\)](#)
- ▲ [Recommendation ITU-T X.1524 CWE, ITU-T CYBEX Series](#)

<http://cwe.mitre.org/>

# CWE/SANS Monster mitigations



## Monster Mitigations

These mitigations will be effective in eliminating or reducing the severity of the Top 25. These mitigations will also address many weaknesses that are not even on the Top 25. If you adopt these mitigations, you are well on your way to making more secure software.

A [Monster Mitigation Matrix](#) is also available to show how these mitigations apply to weaknesses in the Top 25.

ID	Description
<a href="#">M1</a>	Establish and maintain control over all of your inputs.
<a href="#">M2</a>	Establish and maintain control over all of your outputs.
<a href="#">M3</a>	Lock down your environment.
<a href="#">M4</a>	Assume that external components can be subverted, and your code can be read by anyone.
<a href="#">M5</a>	Use industry-accepted security features instead of inventing your own.
<a href="#">GP1</a>	(general) Use libraries and frameworks that make it easier to avoid introducing weaknesses.
<a href="#">GP2</a>	(general) Integrate security into the entire software development lifecycle.
<a href="#">GP3</a>	(general) Use a broad mix of methods to comprehensively find and prevent weaknesses.
<a href="#">GP4</a>	(general) Allow locked-down clients to interact with your software.

See the [Monster Mitigation Matrix](#) that maps these mitigations to Top 25 weaknesses.

Source: <http://cwe.mitre.org/top25/index.html>

# Program Building Blocks – Using Components



- Do not try to develop everything yourself
- Even if you succeed in creating it, you must maintain it!

# Design Patterns



- *Design Patterns: Elements of Reusable Object-Oriented Software* (1994), Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides
- The book describes 23 classic software design patterns
- [https://en.wikipedia.org/wiki/Design\\_Patterns](https://en.wikipedia.org/wiki/Design_Patterns)
- Ideas of patterns precede this book, but became a more popular subject

# Common constructs



- Programs exhibit the same patterns, some examples:
- Solve problems in the same domains
- Need to store lists of strings / characters etc.
- Data structures becomes useful in other programs
- Sorting routines needed in many programs

# Exercise



Now lets do the exercise

## Django String Handling 20min

which is number **47** in the exercise PDF.

## Actual code – JuiceShop and Django 4



# django

Lets look at some code, some may be useful for the exam project:

- JuiceShop source code – how is the login box implemented, can it be improved
- Django email validation – how is the validation done, can it be copied

# Exercise



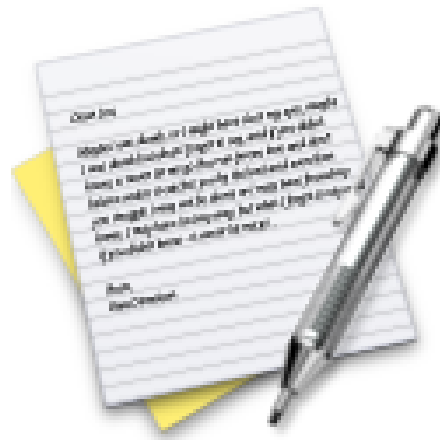
Now lets do the exercise

## JuiceShop Login 15 min

which is number **48** in the exercise PDF.



# Exercise



Now lets do the exercise

## Django email validation 30 min

which is number **49** in the exercise PDF.



## Objectives of the report

This report shall promote the discussion on how privacy by design can be implemented with the help of engineering methods. It provides a basis for better understanding of the current state of the art concerning privacy by design with a focus on the technological side.

“Personal data” means any information relating to an identified or identifiable natural person—for a detailed discussion see [19]. This is related to the term personally identifiable information (PII), as e.g. used in the privacy framework standardised by ISO/IEC [125].

Source: Privacy and Data Protection by Design – from policy to engineering

<https://www.enisa.europa.eu/publications/privacy-and-data-protection-by-design>

- Next, if the application is secure, what about handling and protecting personal data
- As a rule, do not collect, store, process unnecessary data

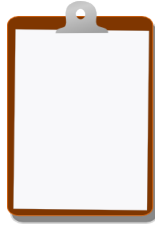
# ENISA: List of Recommendations



- Policy makers need to support the development of new incentive mechanisms for privacy-friendly services and need to promote them.
- The research community needs to further investigate in privacy engineering, especially with a multidisciplinary approach. This process should be supported by research funding agencies. The results of research need to be promoted by policy makers and media.
- Providers of software development tools and the research community need to offer tools that enable the intuitive implementation of privacy properties.
- Especially in publicly co-founded infrastructure projects, privacy-supporting components, such as key servers and anonymising relays, should be included.
- Data protection authorities should play an important role providing independent guidance and assessing modules and tools for privacy engineering.
- Legislators need to promote privacy and data protection in their norms.
- Standardisation bodies need to include privacy considerations in the standardisation process.
- Standards for interoperability of privacy features should be provided by standardization bodies.

Source: ENISA Privacy and Data Protection by Design – from policy to engineering

## For Next Time



Think about the subjects from this time, write down questions

Check the plan for chapters to read in the books

Visit web sites and download papers if needed

Retry the exercises to get more confident using the tools