FROM ZERO TO HERO IN WEB SECURITY RESEARCH

research.checkpoint.com

Check Point
SOFTWARE TECHNOLOGIES LTD

Who are we?

cp<r>

CHECK POINT RESEARCH

Dikla Barda

Security Researcher
Check Point

Yaara Shriki

Security Researcher
Check Point

research.checkpoint.com

Check Point
SOFTWARE TECHNOLOGIES LTD

cp<r>

CHECK POINT RESEARCH

Roman Zaikin

Security Researcher
Check Point

Oded Vanunu

Head Of Products
Vulnerability Research
Check Point

research.checkpoint.com

Check Point
SOFTWARE TECHNOLOGIES LTD


LG Electronics
cp<r>
CHECK POINT RESEARCH
ebay
FORTNITE
AliExpress
skype™
PayPal
ATLASSIAN
TikTok
alexa

Training Overview

cp<r>
CHECK POINT RESEARCH

- You will learn OWASP top 10 in depth and more.
- We will show you everything from both attacker and defender perspective.
- You will practice everything in a lab based on our researches.
- Have Fun!

Case Study (How research is done)

cp<r>
CHECK POINT RESEARCH

Facebook Research

distributing malware through images

Test environment configuration

In this workshop you will practice on a lab that you will setup on your computer.

By using this labs you will learn all the hacking tools and techniques.

https://github.com/romanzaikin/defcon-training

(will be upload before the session)

Don’t have time to setup? Use our server at **https://rzcyber.com/**

Penetration Testing Types

There are 3 types of penetration testing:

White Box Testing – Every detail regarding the project is provided by the company and sometimes even the source code.

Gray Box Testing – Only specific details are provided, the tester have to find the other details by his own.

Black Box Testing – No details are provided.

Research

cp<r>
CHECK POINT RESEARCH

Research is different from penetration testing in the depth of work.

A pentester will try to find bugs as fast as he can and get a POC working.

A researcher will focus more on why things works the way the work and try to tackle with them.

Burp Suite

Burp Suite is an integrated platform for performing security testing of web applications. from initial mapping and analysis of an application's attack surface, through to finding and exploiting security vulnerabilities.

https://portswigger.net/burp/communitydownload

Burp Suite

cp<r>
CHECK POINT RESEARCH

Using burp suite as HTTP Proxy and manipulate the traffic in order to find Web Security Vulnerabilities.

- HTTP Proxy
- Intruder
- Repeater
- Extender
- Comparer
- Scanner
- Decoder
- Spider

OWASP TOP 10

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

https://owasp.org/www-project-top-ten/

A10 Insufficient Logging & Monitoring

Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident. Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.

Path and Information Disclosure

cp<r>
CHECK POINT RESEARCH

An information exposure is the intentional or unintentional disclosure of information to an actor that is not explicitly authorized to have access to that information.

- Errors/debug information (stack trace and so)
- Server headers (server, x-powered-by)
- Path Disclosure (sometimes error disclosure path)
- Admin panels (phpMyAdmin, any admin panel)

A10 - Challenge

Solve A10 – stage1 challenge

1. Try to find which programming language is used in this system?
2. Try to cause the server to crash and show you an stack trace.

Happy Hacking ☺

A10 - Is the Application Vulnerable?

- Auditable events, such as logins, failed logins, and high-value transactions are not logged.
- Warnings and errors generate unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by DAST tools (such as OWASP ZAP) do not trigger alerts.
- The application is unable to detect, escalate, or alert for active attacks in real time or near real time.

A10 - How to Prevent

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts.
- Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.
- Ensure that no stack trace or other sensitive information is disclosed to the attacker.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.
- Establish or adopt an incident response and recovery plan.

A9 - Using Components with known vulnerabilities

cp<r>
CHECK POINT RESEARCH

It’s estimated that well over 80% of all software includes, at least, some open source components. As a result, because of their widespread use, third-party components make a tempting target for potential hackers.

Developers not always updating their code for fear of breaking some features or functionality and so the outdated code stays there.

A9 - Challenge

cp<r>
CHECK POINT RESEARCH

Solve A9 – stage1 challenge

Try to find the vulnerable package used and exploit it by reading the file "/etc/passwd"

Happy Hacking ☺

A9 - Is the Application Vulnerable?

cp<r>
CHECK POINT RESEARCH

- If you do not know the versions of all components you use, includes components you directly use as well as nested dependencies.
- If software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.

A9 - How to Prevent?

- Remove unused dependencies, unnecessary features, components, files, documentation and always update your systems.
- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc. Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component and follow your products security news letters.

A8 - Insecure Deserialization

Insecure Deserialization is a vulnerability which occurs when untrusted data is used to abuse the logic of an application by executing methods from different objects or even execute arbitrary code upon it being **deserialized**.

- Serialize
- Unseserialize

A8 - Demonstration

A8 Demonstration of an insecure deserialization in a php system.

Happy Hacking ☺

A8 - Is the Application Vulnerable?

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker. This can result in two primary types of attacks:

Serialization may be used in applications for:

- Different web services
- Databases, cache servers, file systems
- Store object states

A8 - How to Prevent?

The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.

<u>If that is not possible, consider one of more of the following:</u>

- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.
- Isolating and running code that deserializes in low privilege environments when possible.
- Log deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.

A7 - Cross-Site Scripting

cp<r>
CHECK POINT RESEARCH

Cross-site Scripting (XSS) is a code injection technique. That allows the attacker to execute scripts in a web browser of the victim by including malicious code in a legitimate web page or web application.

- **Persistent/Stored XSS**, where the malicious string originates from the website's database.
- **Reflected XSS**, where the malicious string originates from the victim's request.
- **DOM based XSS**, where the vulnerability is in the client-side code rather than the server-side code.

* **Blind XSS,** when there is no direct reflection from the server but your script is executed in the application or in another application somewhere.

Case Study (XSS in DJI)

cp<r>
CHECK POINT RESEARCH

DJI - Drone
(XSS Demo)

https://research.checkpoint.com/2018/dji-drone-vulnerability/

A7 - Challenge

Solve Challenges 1 to 6.

Happy Hacking ☺

A7 - Cross-Site Scripting

In XSS we search for code reflections, for example if we insert “aaa” we search for those characters in the response.

To protect against XSS we use encoding:

```
const encode = require('html-entities').encode;
encode(req.query.search)
```

A7 - Cross-Site Scripting

cp<r>
CHECK POINT RESEARCH

Well known HTML entities

Result	Description	Entity Name	Entity Number
	non-breaking space		
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	double quotation mark	&quot;	&#34;
'	single quotation mark (apostrophe)	&apos;	&#39;

Case Study (Open Redirect Example)

cp<r>
CHECK POINT RESEARCH

AliExpress
(open redirect)

https://research.checkpoint.com/2017/christmas-coming-criminals-await/

A7 - Cross-Site Scripting

Solve Challenges 7.

Happy Hacking ☺

Content Security Policy

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement and distribution of malware.

A7 - Cross-Site Scripting

Solve Challenges 8.

Happy Hacking ☺

A7 - How to Prevent?

- Using frameworks that automatically escape XSS by design.
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities.
- Applying context-sensitive encoding when modifying the browser document on the client side acts against DOM XSS.
- Enabling a Content Security Policy (CSP) as a defense-in-depth mitigating control against XSS.

A6 - Security Misconfiguration

cp<r>
CHECK POINT RESEARCH

These vulnerabilities often occur due to insecure default configuration, poorly documented default configuration, or poorly documented side-effects of optional configuration. This could range from failing to set a useful security header on a web server, to forgetting to disable default platform functionality that could grant administrative access to an attacker.

CORS

Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any other origins (domain, scheme, or port) than its own from which a browser should permit loading of resources.

http://rzcyber.com:443

Scheme :// domain : port

Same Origin Policy (SOP)

The **same-origin policy** is a critical security mechanism that restricts how a document or script loaded by one origin can interact with a resource from another origin.

It helps isolate potentially malicious documents and reducing possible attack vectors.

Same Origin Policy (SOP) Rules

The following table gives examples of origin comparisons with the URL `http://store.company.com/dir/page.html`:

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner/another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

A6 - Challenge

Solve A6 – stage1 challenge

Try to abuse the CORS and steal the email and username of the victim by sending an AJAX from a different host

Happy Hacking ☺

Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.

If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth.

If the victim is an administrative account, CSRF can compromise the entire web application.

A6 - Challenge

Solve A6 – challenges stage2 – stage4

Try to inject the message "hacked" to the forum via "Cross Site Request Forgery" vulnerability in all 3 challenges (2,3,4)

Happy Hacking ☺

A6 - Is the Application Vulnerable?

The application might be vulnerable if the application is:

- Unnecessary features are enabled or installed
- Default accounts and their passwords still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, latest security features are disabled or not configured securely.
- The server does not send security headers or directives or they are not set to secure values.
- The software is out of date or vulnerable (see **A9:2017-Using Components with Known Vulnerabilities**).

A6 - How to Prevent?

Secure installation processes should be implemented, including:

- A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down.
- Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to setup a new secure environment.
- A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.

Case Study (**Demo in Amazon Alexa**)

cp<r>
CHECK POINT RESEARCH

AMAZON ALEXA RESEARCH

https://research.checkpoint.com/2020/amazons-alexa-hacked/

A5 - Broken Access Control

Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others. These checks are performed after authentication, and govern what 'authorized' users are allowed to do.

Parameter tampering

Parameter tampering is a form of Web-based attack in which certain parameters in the (URL) or Web page form field data entered by a user are changed without that user's authorization.

- Post id
- isAdmin
- Application isolation

LFI/RFI/SSRF

cp<r>
CHECK POINT RESEARCH

Local File Inclusion is the process of including files, that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application.

Remote File Inclusion are vulnerabilities that often attack PHP websites, allowing attackers to include remotely hosted files.

Server-Side Request Forgery the attacker can abuse functionality on the server to read or update internal resources on the server.

A5 - Challenge

Solve A5 – challenges stage1 – stage2

1. On stage1 Try to read /etc/passwd via "Local File Inclusion" vulnerability.
2. On stage2 try to find another application on the server and access it’s secret api at /secret

Happy Hacking ☺

A5 - Is the Application Vulnerable?

cp<r>
CHECK POINT RESEARCH

- Bypassing access control checks by modifying the URL, internal application state, or the HTML page, or simply using a custom API
- Allowing the primary key to be changed to another's users record, permitting viewing or editing someone else's account.
- Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token or a cookie or hidden field manipulated to elevate privileges, or abusing JWT invalidation.
- CORS misconfiguration allows unauthorized API access.

A5 - How to Prevent?

- Implement access control mechanisms once and re-use them throughout the application, including minimizing CORS usage.
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update, or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots.
- Log access control failures, alert admins when.
- JWT tokens should be invalidated on the server after logout.

A4 - XML External Entities (XXE)

Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations.

Case Study (**XXE Demo in Android Studio**)

cp<r>
CHECK POINT RESEARCH

XXE Android Studio - ParseDroid

https://research.checkpoint.com/2017/parsedroid-targeting-android-development-research-community/

A4 - Challenge

Solve A4 – challenge stage1

Try to read /etc/passwd via "XML External Entities" vulnerability.

Happy Hacking ☺

A4 - Is the Application Vulnerable?

The application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor.

- Any of the XML processors in the application or SOAP based web services has document type definitions (DTDs) enabled.
- If the application uses SAML for identity processing within federated security or single sign on (SSO) purposes. SAML uses XML for identity assertions, and may be vulnerable.
- Being vulnerable to XXE attacks likely means that the application is vulnerable to denial of service attacks including the Billion Laughs attack

A4 - How to Prevent?

- Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data.
- Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system.
- Disable XML external entity and DTD processing in all XML parsers in the application, as per the OWASP Cheat Sheet 'XXE Prevention'

A3 - Sensitive Data Exposure

cp<r>
CHECK POINT RESEARCH

Rather than directly attacking crypto, attackers steal keys, or steal clear text data off the server, Developers sometimes forget test cases or other files on production servers.

- Git
- Certificates
- Logs with passwords

A3 - Challenge

Solve A3 – challenge stage1

Try to leak the code from the application somehow.

Happy Hacking ☺

A3 - Is the Application Vulnerable?

cp<r>
CHECK POINT RESEARCH

- Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, and FTP. External internet traffic is especially dangerous. Verify all internal traffic.
- Are any old or weak cryptographic algorithms used either by default or in older code?
- Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?
- Is encryption not enforced, e.g. are any user agent (browser) security directives or headers missing?
- Does the user agent (e.g. app, mail client) not verify if the received server certificate is valid?

A3 - How to Prevent

- Classify data processed, stored or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Apply controls as per the classification.
- Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).

A2 - Broken Authentication

Authentication and session management includes all aspects of handling user authentication and managing active sessions. While authentication itself is critical aspect to secure, even solid authentication mechanisms can be undermined by flawed credential management functions, including password change, "forgot my password", "remember my password", account update, and other related functions.

Case Study (**Auth0 Bypass in LG**)

cp<r>
CHECK POINT RESEARCH

LG Broken Authentication

https://blog.checkpoint.com/2017/10/26/homehack-how-hackers-could-have-taken-control-of-lgs-iot-home-appliances/

A3 - Challenge

Solve A2 – challenge stage1

Try to abuse the login mechanism and login to the user "roman".

Happy Hacking ☺

A2 - Is the Application Vulnerable?

cp<r>
CHECK POINT RESEARCH

- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin“.
- Uses weak or ineffective credential recovery and forgot-password processes.
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL.
- Does not rotate Session IDs after successful login.
- User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

A2 - How to Prevent?

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login.

A1 - Injection

Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.

SQL Injection

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

SELECT * FROM username=""**or 1=1 – "** WHERE password=""

NoSQL Injection

cp<r>
CHECK POINT RESEARCH

- NoSQL databases do not use SQL to perform queries.
- While NoSQL databases (like MongoDB) do not use SQL for queries, they still do queries based upon user input.
- This means that they are still vulnerable to injection attacks if the developer does not properly perform input sanitization.

A1 - Is the Application Vulnerable?

cp<r>
CHECK POINT RESEARCH

- User-supplied data is not validated, filtered, or sanitized by the application.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.
- Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP and more.

A1 - How to Prevent?

- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface.
- Even when parameterized, stored procedures can still introduce SQL.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

Resources

Github: https://github.com/romanzaikin/defcon-training

Labs: https://rzcyber.com

CPR Team:

Oded Vanunu, Roman Zaikin, Dikla Barda, Yaara Shriki


cp<r>
CHECK POINT RESEARCH
THANK YOU
research.checkpoint.com
Check Point
SOFTWARE TECHNOLOGIES LTD