Protect Against Session Cookie Attacks

- Mitigation using best practices and tools -

Introduction

When it comes to IT Security, hardening the high-risk areas or patching the gray areas is not as easy as: "a walk in the park". The last decade's approach to IT security has been too loose the least, if not, NON-EXISTENT!

The booming industry of cloud computing in conjunction with the infinite web-based solutions over the Internet shined the path to an era where personal, medical and business data are at stake if not carefully handled and secured.

The scope of this paper is to educate and inform IT Security professionals of a common, yet not always effectively treated, attack and how to ensure that such security breaches either affect a small surface area or rapidly diminished.

Session cookie attacks/code execution & abuse

The session cookie attack is a common case where an attacker manages to steal a cookie (compromises) from the user's browser and act as the original user, thus having full access and control over his account without the need to know the actual credentials. Furthermore, the attacker can also change the credentials, verification emails and other data and thus block the user out forever!

Such kind of attacks also enable remote code execution (if an API is available) and possible abuse of the whole system and not just of one account. More specifically, the attacker, can cause irreversible damage with:

- Distributed DoS
- Industrial Espionage
- Information Theft
- API Abuse
- Data Corruption
- Impersonation
- Busines Intelligence Data Devalidation

So, the session cookie should NOT be the only way to authenticate and continue trusting the user throughout the session lifetime! I go into more detail on the subject in this paper as I unravel the logic and methodologies behind the solution.

Recently I have discovered such an exploit for Cisco UMBRELLA as part of my security research. You can check the official report here.

A secure framework (By design)
Development has to be secure by design, hence the efforts the last few years for more complete Web platforms, CMS, etc. with integrated security that follows best practices and principles. In fact, back in 2009 and 2011 as part of my MSc at Kingston University, I released a paper regarding the right tools for scanning and discovering vulnerabilities and later my thesis on how to design and implement a secure web application.

In my spare time I've been working also – and still do, to deliver a simple, yet powerful, FREE and OPEN SOURCE solution that provides a fundamental framework for secure and easy development.

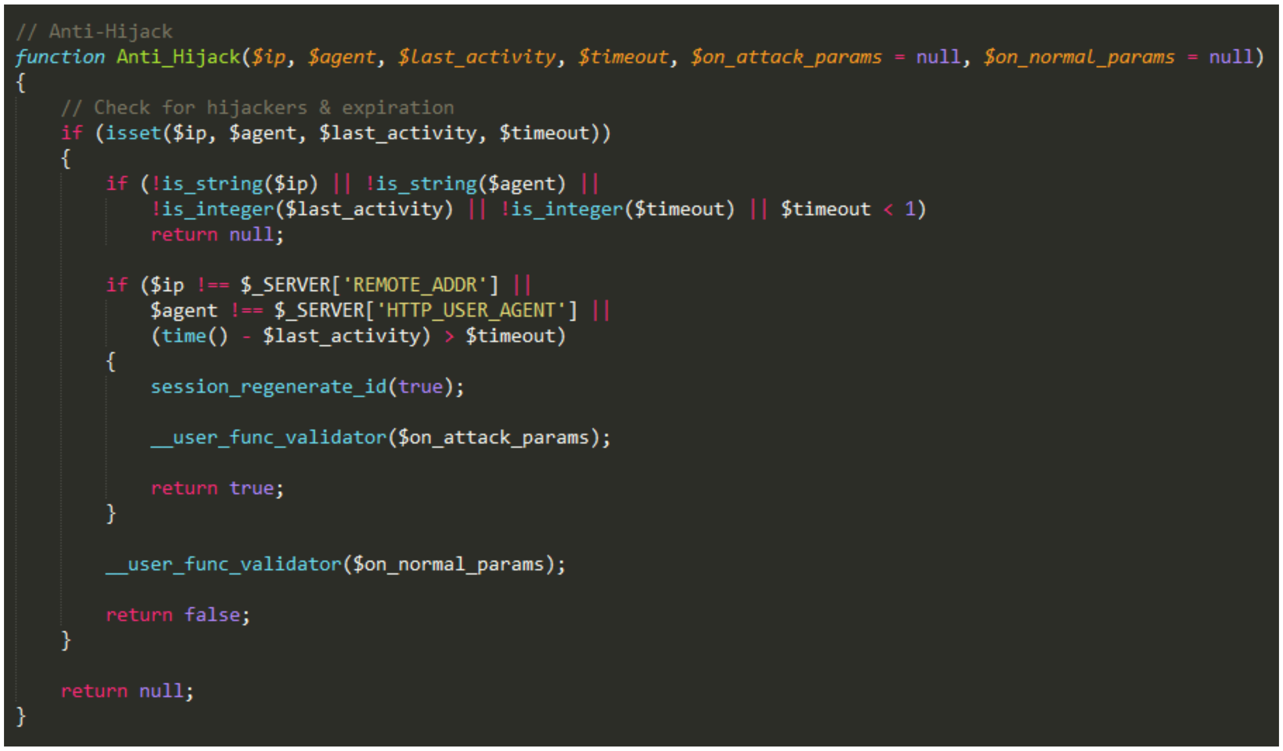
My efforts lead to: micro-MVC. An agile, small, productive, robust and secure by design MVC framework for PHP (with integrated AJAX support). You may learn more on the philosophy behind its development, clone it, contribute or just download it from GitHub at: https://github.com/g0d/micro-MVC.

Anti-Hijack (Introduction)
It's not easy to provide a super complete, always working and smart solution. Especially in large enterprise applications (e.g. Cisco UMBRELLA, CloudFlare, etc.) the flaws can be extremely difficult to be spotted and remediation usually takes a while. However, a proactive solution that – at least – utilizes common techniques and useful methodologies to minimize the impact of an attack is required.

The micro-MVC framework comes with such a utility. Anti-Hijack is a small PHP extension that tries to limit the possibility of a session cookie attack or the abuse of any services you provide to your users. You can check the video about the basic mechanics behind its implementation or you can continue reading through as I explain, in short, the key features.

Anti-Hijack (How it works)
Anti-Hijack is a relatively easy to comprehend function. It takes under consideration fundamental principles of how HTTP/S and cookies work and tries to minimize the risk. Its effectiveness though is not guaranteed. It takes several tests, server-side configurations and an experienced programmer, with a good understanding of IT Security principles possibly, to utilize it to the maximum.

The full function specification is as follows:

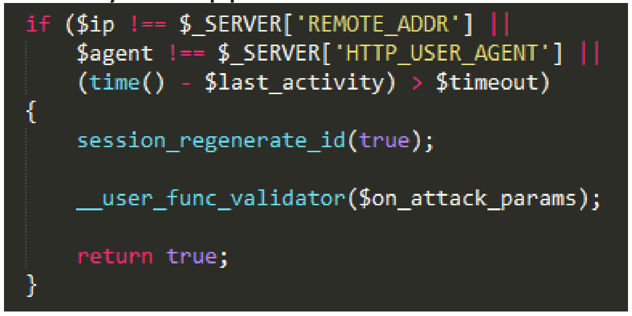


```php
// Anti-Hijack
function Anti_Hijack($ip, $agent, $last_activity, $timeout, $on_attack_params = null, $on_normal_params = null)
{
    // Check for hijackers & expiration
    if (isset($ip, $agent, $last_activity, $timeout))
    {
        if (!is_string($ip) || !is_string($agent) ||
            !is_integer($last_activity) || !is_integer($timeout) || $timeout < 1)
            return null;

        if ($ip !== $_SERVER['REMOTE_ADDR'] ||
            $agent !== $_SERVER['HTTP_USER_AGENT'] ||
            (time() - $last_activity) > $timeout)
        {
            session_regenerate_id(true);

            __user_func_validator($on_attack_params);

            return true;
        }

        __user_func_validator($on_normal_params);

        return false;
    }

    return null;
}
```

A very straightforward explanation can be found in this video. The absolutely essential arguments of this function are the:

1. User/Client IP
2. Agent (Browser info)
3. Last Activity (Time in UNIX epoch since the last request)
4. Timeout (Time in [milli]seconds/minutes/hours after the session cookie expires)

The core code that protects your application and infrastructure is:



```php
if ($ip !== $_SERVER['REMOTE_ADDR'] ||
    $agent !== $_SERVER['HTTP_USER_AGENT'] ||
    (time() - $last_activity) > $timeout)
{
    session_regenerate_id(true);

    __user_func_validator($on_attack_params);

    return true;
}
```

The code snippet shows three basic comparisons:

1. The user IP has to match that of the session cookie currently in use by the web server and PHP.
2. The user agent must be the same (browser, version number, OS version and name) with that of the session initiator.
3. The timeout ensures that an active request (GET, POST) via form posts, AJAX, etc. will be received and certify that the user agent is still alive.

The "magic" line is where we call the: *session_regenerate_id(true);*
Essentially this line tells to PHP to discard the currently active session ID and delete all related memory caches and connections with the user's browser.

Beware though! This does not ensure that the web server nor the browser will immediately regard and honor the request. An automated attack tool (a script) needs just a short period of time (a few milliseconds really) to use this cookie and take control of your web application or cause a [D]DoS. This is more common to large enterprise applications with virtual servers and technology stacks that utilize containers. Since those systems are not mission critical and definitely not time sensitive this can lead, relatively easy, to a side channel attack!

Thankfully, micro-MVC comes with out-of-the-box enterprise-ready configurations for Apache and NGINX. Finally, micro-MVC has an integrated supervising dispatcher that controls the web access and routing flow in a convenient and secure way. The dispatcher hides critical information from the attacker and also denies access to malicious operations in a transparent way.

Conclusions

No application is 100% secure and probably never will be!

This doesn't mean though that we must leave anything to luck or just accept the fact that someday we will be hacked or attacked. Instead, we enforce better policies, work on the basic principles of ISO27001/27002 and apply GDPR (if you live in the EU) when possible.

Finally, it is strongly recommended that we do pen-tests every 3 to 6 months and re-assess our business infrastructure. Risk analysis & management is not a luxury nor a fancy trend ("a thing") anymore. It is an essential and useful tool that ensures stability and business continuity!

George Delaportas
Enterprise Architect & IT Strategist
Certified Hacker & Researcher (ViR4X)
IT Director at VEDICOR S.A.
https://www.linkedin.com/in/gdelaportas
https://kingston.academia.edu/GeorgeDelaportas
https://github.com/g0d