Exploits, Vulnerabilities and Payloads: Practical Introduction

infosecmatter.com/exploits-vulnerabilities-and-payloads-practical-introduction

May 22, 2020

As penetration testers, we have to use exploits very often to demonstrate vulnerabilities to our customers. But where do we get them? How do we use them? And what should we keep in mind when using them? Exploits are dangerous little pieces of code and therefore a healthy dose of caution is always a good practice.

Exploit vs. vulnerability

In the infosec industry, we can sometimes witness long debates about what exactly a vulnerability, an exploit or a software bug is and where one term begins and the other ends.

Let's not complicate things too much here and let's just focus on the essentials.

Software bugs are where it all begins. They are simply programming errors and they are usually very well defined and named. Of course they don't have to be just in software – they can be in hardware as well.

Here are some examples of software bugs:

- Buffer overflow
- Race condition
- Access violation
- Infinite loop
- Division by zero
- Off-by-one error
- Null pointer dereference
- Input validation error
- Resource leak

Software bugs can lead to vulnerabilities. Not always, but sometimes they do.

Vulnerabilities are somewhat exact, but sometimes have funny names. Vulnerabilities are basically software bugs which can be taken advantage of to achieve an unintended or unanticipated behavior.

Here are some examples of vulnerabilities:

- BlueKeep
- Shellshock
- Dirty COW
- Heartbleed

- EternalBlue
- SQL injection
- Code injection
- Directory traversal
- XSS, CSRF, SSRF

And then there's the **Risk** (Threat) which may materialize, if someone decides to take advantage of a vulnerability and exploit it.

Here are some examples of security risks:

- Remote code execution
- Authentication bypass
- Sensitive information disclosure
- Denial of Service
- Privilege escalation
- Security feature bypass
- User session takeover
- Malicious file upload
- Man-in-the-middle

Here's how software bugs, vulnerabilities and security risks come together:

Software bug examples:	→	**Vulnerability** examples:	→	**Risk / Threat** examples:
• Buffer overflow		• BlueKeep		• Remote code execution
• Race condition		• Shellshock		• Authentication bypass
• Access violation		• Dirty COW		• Sensitive information disclosure
• Infinite loop		• Heartbleed		• Denial of Service
• Division by zero		• EternalBlue		• Privilege escalation
• Off-by-one error		• SQL injection		• Security feature bypass
• Null pointer dereference		• Code injection		• User session takeover
• Input validation error		• Directory traversal		• Malicious file upload
• Resource leak		• XSS, CSRF, SSRF		• Man-in-the-middle

infosecmatter.com

An **exploit** would then represent an actual conjunction between the three terms, materializing the risk into reality.

In definition, an exploit is a piece of code, a program or a carefully crafted data which takes advantage of a vulnerability to achieve an unintended or unanticipated behavior (materializing the risk) in the software that contains a bug.

Go back to top.

Local vs. Remote

One way the exploits are categorized in the infosec industry is by dividing them into 2 major groups – local and remote exploits.

Local exploits are codes with purpose to exploit a vulnerability locally on the system where we already have (limited) access.

These exploits are almost always privilege escalation exploits with the aim to increase our privileges to a higher (ideally administrative / root / ring 0) level so that we would have a complete control over the resources of the target system.

Remote exploits are codes with purpose to exploit a vulnerability on a remote system without having any prior access to it.

These exploits are typically targeted against a specific network service (of a specific vulnerable version). The objective is to obtain an unauthorized access to information or obtain access to the remote system's resources.

There are many different types of exploits. Let's see how the infosec industry categorizes them further.

Exploit types

This section lists the most common types of exploits that we can come across in our daily infosec life.

Denial of Service (DoS)

DoS is an exploit causing a crash of a given service / system.

During professional penetration tests there is almost never a reason to use a DoS exploit.

In fact, causing a crash or an unavailability of a system is highly unwanted outcome. This can lead to a business downtime and consequently a very unpleasant conversations with the client. At minimum.

But it could also be much worse – there could even be revenue losses, losses of productivity and other financial consequences for which we or our employer could be held liable. Make sure to have the paperwork in check before doing any pentest.

In any case, we should always stay away from using DoS exploits. There is nothing to be gained by using them anyway. Unless of course it is explicitly desired and authorized by the customer.

Local Privilege escalation (LPE)

As mentioned above, privilege escalation exploits aim to exploit vulnerabilities locally on the system where we already have (limited) access.

The objective is to obtain administrative privileges, typically:

- "Domain Admin" in Active Directory
- "NT Authority\System" on Windows systems

- “Root” user on UNIX like systems

But, there are also many other ways how to escalate privileges – not only by using exploits. Here’s one of the best resources on privilege escalation tactics:

> https://attack.mitre.org/tactics/TA0004/

Remote Code Execution (RCE)

These are the most popular exploits. They allow us to execute arbitrary code on the target system.

Sometimes, however, exploits can cause a crash of the target.

One example would be the infamous EternalBlue (aka. MS17-010) vulnerability. There are many MS17-010 exploits and some of them are of a poor quality, causing a crash of the entire operating system.

Therefore, we should be always vigilant about using RCE exploits. Sometimes, clients may even ask us to notify them before doing any active exploitation.

In fact, it is quite typical in most mature environments for the client to ask us to refrain from doing any exploitation without obtaining authorization first.

Web Applications (WebApps)

There’s a whole cluster of exploits targetting vulnerabilities in web applications and web technologies in general.

These include vulnerabilities such as:

- Directory Traversal
- SQL Injection (SQLi)
- Authentication Bypass
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Server-Side Request Forgery (SSRF)
- XML External Entity Injection (XXE)
- Local / Remote File Inclusion (LFI / RFI)
- And many more..

As the whole world shifts towards web technologies, WebApp exploits make up the highest portion of published exploits by far.

Client-Side exploits

Client-side exploits typically exploit vulnerabilities in client applications such as:

- PDF viewers

- Web browsers
- Chat / IM / Email clients
- FTP, SSH, DHCP clients
- And many more..

This type of exploits are used very rarely during a penetration test, if ever. Thus, we will not cover them here in much detail.

Let's just mention that not even our pentesting tools, packet sniffers or scanners are spared from having vulnerabilities (and exploits).

Proof of Concept (PoC) exploits

In the world of exploits, we can often times come across a PoC exploit. What is a PoC exploit?

Turns out it can be anything! Unfortunately, the infosec industry doesn't follow any strict guidelines on what a PoC exploit should or should not be.

PoC exploit could be a test code to a check whether a particular vulnerability exists on a given system or not (without exploiting it).

But it could also be a "dirty" or an "unstable" version of a fully potent exploit!

We never know. Therefore, we should always treat PoC exploits as any other exploit – a "PoC" is simply a synonym for "exploit".

Go back to top.

Where to find exploits?

The following sections provide a list of generally accepted and recommended sources of exploits.

Exploit Database

The Exploit Database (https://www.exploit-db.com/) operated and maintained by Offensive Security is one of the best places where to get good quality exploits. It is an archive of public exploits and security advisories.

There are currently over 42,729 exploit entries in their database at the time of writing this article and the number grows practically every day.

There is also a command line utility **searchsploit** for accessing the exploit archive. This utility is pre-installed on Kali Linux by default and it is also available for other systems as well.

Here's an example of using the utility to search for JBoss exploits:

```
searchsploit jboss
```

```
kali@kali:~$ searchsploit jboss
------------------------------------------------------------------ ----------------------------
 Exploit Title                                                    |  Path
------------------------------------------------------------------ ----------------------------
Apache Tomcat/JBoss EJBInvokerServlet / JMXInvokerServlet (RMI over HT | php/remote/28713.php
Cisco DCNM JBoss 10.4 - Credential Leakage                        | java/remote/47885.txt
Cisco Security Monitoring Analysis and Response System JBoss - Command | hardware/remote/28245.pl
Cisco/Protego CS-MARS < 4.2.1 - 'JBoss' Remote Code Execution     | hardware/remote/2048.pl
JBoss & JMX Console - Misconfigured Deployment Scanner            | jsp/webapps/17924.pl
JBoss - DeploymentFileRepository WAR Deployment (via JMXInvokerServlet | multiple/remote/21080.rb
JBoss - Java Class DeploymentFileRepository WAR Deployment (Metasploit | multiple/remote/16316.rb
JBoss 3.0.8/3.2.1 - HSQLDB Remote Command Injection               | multiple/remote/23221.txt
JBoss 3.x/4.0.2 - HTTP Request Remote Information Disclosure      | multiple/remote/25842.txt
JBoss Application Server 4.2 < 4.2.0.CP09 / 4.3 < 4.3.0.CP08 - Remote | jsp/webapps/16274.pl
JBoss AS 2.0 - Remote Command Execution                           | windows/remote/17977.txt
JBoss AS 3/4/5/6 - Remote Command Execution                       | multiple/webapps/36575.py
JBoss JMX - Console Beanshell Deployer WAR Upload and Deployment (Meta | multiple/remote/16319.rb
JBoss JMX - Console Deployer Upload and Execute (Metasploit)      | multiple/remote/16318.rb
JBoss JMXInvokerServlet JMXInvoker 0.3 - Remote Command Execution | java/webapps/36553.java
JBoss Remoting 6.14.18 - Denial of Service                        | multiple/dos/44099.txt
JBoss Seam 2 - Arbitrary File Upload / Execution (Metasploit)     | jsp/remote/36653.rb
Red Hat JBoss EAP - Deserialization of Untrusted Data             | java/webapps/40842.txt
Websphere/JBoss/OpenNMS/Symantec Endpoint Protection Manager - Java De | multiple/remote/44552.py
------------------------------------------------------------------ ----------------------------
Shellcodes: No Results
kali@kali:~$
```

The actual exploits can be then found under the /usr/share/exploitdb/exploits/ directory.

Vulners CVE database

The Vulners CVE database (https://vulners.com/search) is another great source of exploits.

It aggregates vast amount of information from many different sources. This allows them to provide details practically on any publicly known vulnerability.

The Vulners database even indexes information from vulnerability scanners such as Nessus or OpenVAS allowing us to search through the contents of their plugins.

This can reveal some very interesting information.

Each entry contains also a reference section with links to the actual advisories and mailing lists. This can also be a great source of information possibly leading to an exploit.

GitHub exploit repositories

There are also numerous GitHub repositories containing exploits and CVE PoC codes.

Here are some of the best exploit repositories:

- https://github.com/offensive-security/exploit-database
- https://github.com/offensive-security/exploit-database-bin-sploits
- https://github.com/qazbnm456/awesome-cve-poc
- https://github.com/Mr-xn/Penetration_Testing_POC
- https://github.com/Medicean/VulApps
- https://github.com/nixawk/labs

- https://github.com/Coalfire-Research/java-deserialization-exploits
- https://github.com/toolswatch/vFeed
- https://github.com/Metnew/uxss-db (Browser vulns)
- https://github.com/tunz/js-vuln-db (JavaScript vulns)

Note that these repositories should be used with utmost caution, because they may contain malicious content.

Make sure to follow safety guidelines described below in the final part of this article.

Exploit frameworks

Exploit frameworks are a big software bundles that allow us to automate variety of penetration testing activities in a standardized way and on a large scale.

They also contain a large number of exploits which are tested and safe to use.

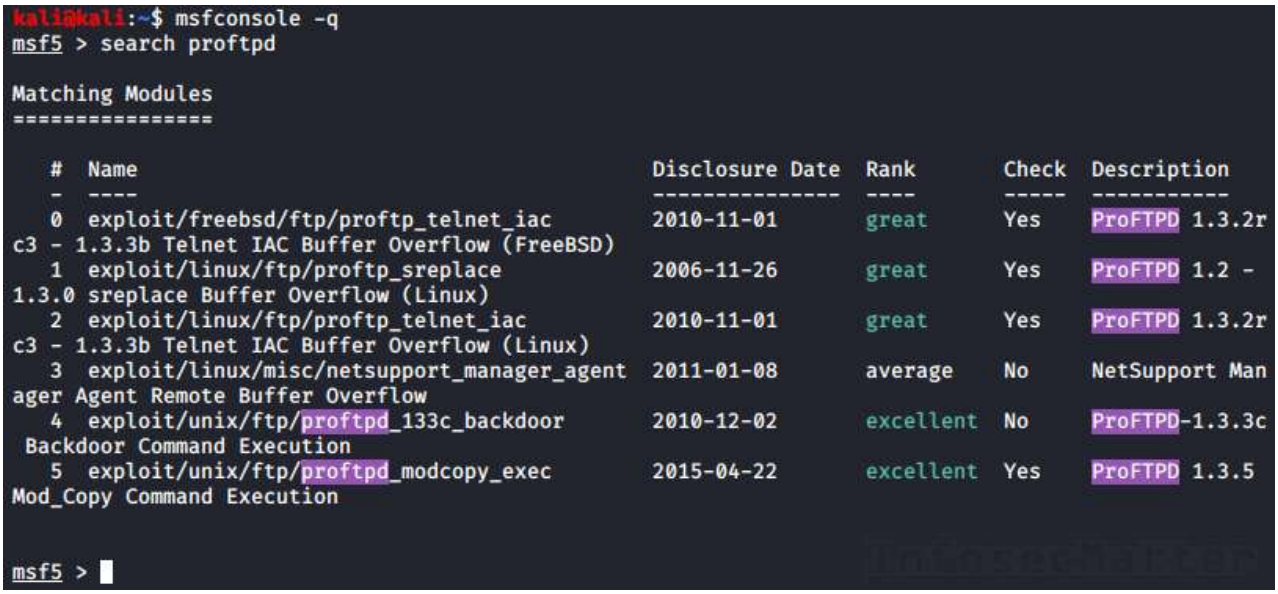
Here are some of the most popular exploit frameworks:

- Cobalt Strike (https://www.cobaltstrike.com/)
- Metasploit Framework (https://www.metasploit.com/)
- Metasploit Pro (https://www.rapid7.com/products/metasploit/)
- CORE IMPACT (https://www.coresecurity.com/core-impact)
- Immunity CANVAS (https://www.immunityinc.com/products/canvas/)
- D2 Elliot Web Exploitation Framework (https://www.d2sec.com/elliot.html)
- BeEF Browser Exploitation Framework (https://beefproject.com/)

The Metasploit Framework is probably the most popular and it is also pre-installed on Kali Linux by default.

Here's an example of searching for ProFTPd exploits in Metasploit:

```
msfconsole
msf5 > search proftpd
```



We could also just as easily search for a CVE.

Exploit frameworks are extremely useful, because they allow us to easily chain the exploitation with the related tasks such as:

- payload / shellcode generation
- pivoting and lateral movement
- post-exploitation tasks

They also contain various scanners and a number of auxiliary modules useful for pentesting.

Exploit frameworks are a huge topic and there are entire books written about them.

IMPORTANT NOTE

Do not confuse exploit frameworks with exploit kits!

Exploit kits are toolkits used by cybercriminals to automate attacks on client end-user machines. They are typically used to distribute malware and to help carry out illegal activities often times involving fraud and theft.

Exploit kits belong to the realm of computer viruses and although they also contain exploits, there is no place for using then during a penetration test.

What is a payload?

Since we are talking about exploits here, we have to mention payloads, because they are inseparable part of most exploits. To understand why exploits have payloads, let me ask you a question:

What happens after we exploit a software bug, let’s say a buffer overflow?

Well, turns out whatever we want! That’s where payloads come into play.

Payloads basically define an action that we want to perform after we exploit a vulnerability. For instance:

- Execute a code
- Spawn a reverse shell
- Create a backdoor
- Create a user
- Read a file

Anything we want.

Payloads are typically written in form of a shellcode, but it is not a rule. For instance, webapp exploits have payloads in a text form.

What is a shellcode?

Shellcodes belongs to the area of binary exploitation. A shellcode is basically a binary form of a payload – a piece of code defining the action (instructions) that we want to execute during the exploitation.

Typically, shellcode is written in a machine code that is appropriate to the target processor architecture and the operating system. For instance:

- x86 / Windows
- Dalvik / Android

Although shellcoding is about coding, we don't really have to code anything – it is completely automated by exploit frameworks such as Metasploit.

Here's an example of the **msfvenom** utility from Metasploit, which can generate payloads practically on every thinkable platform and architecture:

```
msfvenom --list platforms
msfvenom --list archs
```

Msfvenom can generate virtually any payload that we could possibly want while also supporting various obfuscation and encoding methods.

It is a swiss-army knife of payload generation and its features are far beyond the scope of this article.

Here are some online resources with examples on how to use msfvenom:

- https://www.offensive-security.com/metasploit-unleashed/msfvenom/
- https://medium.com/@PenTest_duck/offensive-msfvenom
- https://netsec.ws/?p=331

Go back to top.

How to know which exploit to use?

This typically boils down to a thorough reconnaissance and identifying which particular version of the software is running on our target. Once we got the exact version information, we can then proceed and look for an exploit.

There is also one little trick that we could use..

The **searchsploit** utility described above can parse output from the Nmap scanner and recommend exploits based on the detected versions.

All we have to do is scan our target using Nmap service detection scan (-sV) and save the output in an XML file (-oX). Like this:

```
nmap -sV -oX file.xml <target>
```

Then we can provide the file.xml output to the seachsploit like this:

```
searchsploit --nmap file.xml
```

Go back to top.

How to use exploits safely?

Exploits are a priori dangerous and mishandling them can have disastrous effects.

Because of their seemingly cryptic and obscure nature, they are generally hard to understand. And this creates yet another risk.

For instance, it would be quite easy to sneak some unpleasant surprise into the exploit – a hidden code or a backdoor.

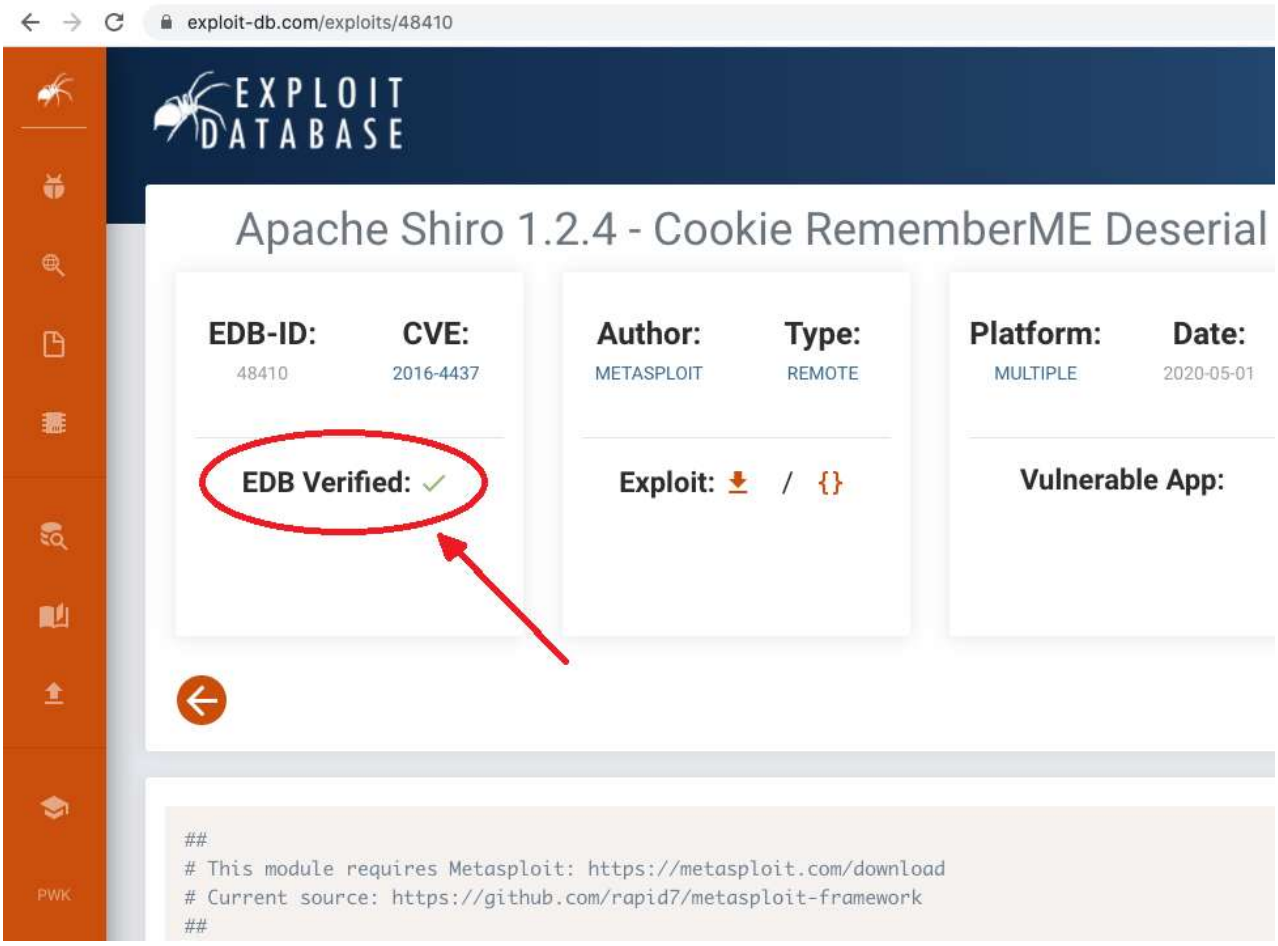
Therefore, an extra dose of caution is always recommended and this section provides some tips on how to minimize those risks.

Verified vs. unverified exploits

Before running any exploit, we should have an idea about what the exploit does.

I'm not suggesting that we should understand in detail how exactly the exploit works, but we should have at least some level of certainty that it doesn't do something more than it claims.

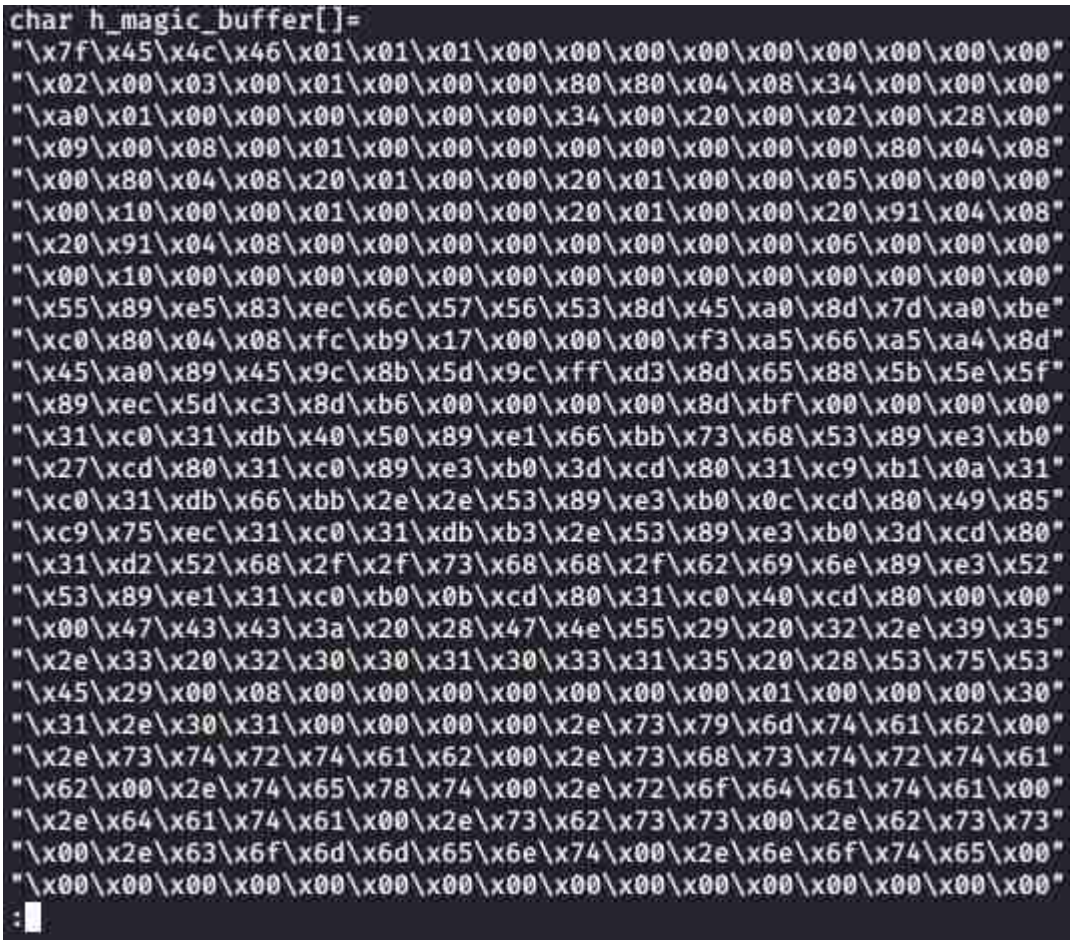
One of the indicators that an exploit is safe is to look for the **'EDB Verified'** mark in the Exploit Database (https://www.exploit-db.com/) web interface:



The green ticks gives indication that the exploit has been tested in the Offensive Security labs and that it is safe to use.

Beware of unknown binary blobs

We should be extra cautious when we see an unknown binary blob in an exploit like in this example:



Why is this buffer there and what does it mean?

Unless we understand assembly language, reverse engineering and binary file analysis, there is not much that we can do other than to test the exploit in a test environment.

But we could still try to analyze it using the following tricks.

See if there are any printable / ASCII characters:

```
echo -en "\x7f\x45\x4c\x46\x01....." | xxd
```

See if it matches any known file type:

```
echo -en "\x7f\x45\x4c\x46\x01....." | file -s -
```

Another way to find whether it contains any known file types (also inside):

```
echo -en "\x7f\x45\x4c\x46\x01....." > blob.bin
binwalk blob.bin
```

Interpret the blob as machine code (instructions) and read the assembly code:

```
echo -en "\x7f\x45\x4c\x46\x01....." | ndisasm -u -
```

If we are unsure, our best shot is to test it in a test environment.

Safety guidelines for working with exploits

Here’s a list of fundamental safety guidelines to keep in mind when using exploits:

- Make sure to use only trustworthy and reputable sources of exploits
- Do not download exploits from private forums or from people you don’t know
- Make an effort to test the exploit before running it on a target
- Always inspect every exploit in a text editor
- Beware of terminal escape injections

Conclusion

In this article we had a peak into the world of exploits. It is a big world with lots of complicated things to learn and know about. But even if we are not binary ninjas, we should still be able to use exploits comfortably and safely during our pentests.

Hopefully this article provided sufficient information on where to get them, how to use them and how to minimize the risks associated with them.