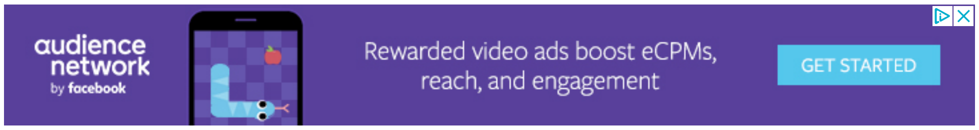
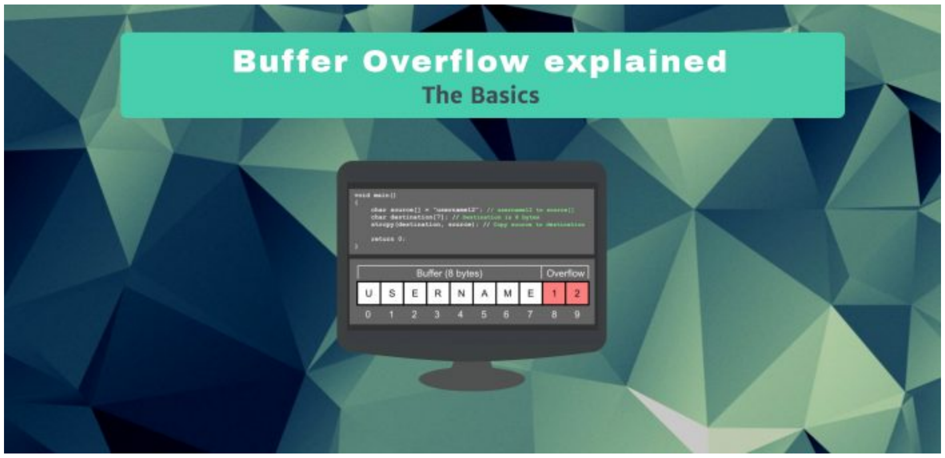
Hacking Tutorials


audience network by facebook
Rewarded video ads boost eCPMs, reach, and engagement
GET STARTED



```
void main()
{
    char source[] = "username12"; // username12 to source[]
    char destination[7]; // Destination is 8 bytes
    strcpy(destination, source); // Copy source to destination

    return 0;
}
```

Buffer overflow explained: The basics

3

BY HACKING TUTORIALS ON JANUARY 2, 2017 EXPLOIT TUTORIALS

One of the most common and oldest security vulnerabilities in software are buffer overflow vulnerabilities. Buffer overflow vulnerabilities occur in all kinds of software from operating systems to client/server applications and desktop software. This often happens due to bad programming and the lack of or poor input validation on the application side. In this article we will look at what a buffer overflow exactly is, how they work and how they can become serious security vulnerabilities. We will also look at what happens when a buffer overrun occurs and mitigation techniques to minimize their harmful effects.

What is a buffer overflow?

A buffer overflow is a situation where a running program attempts to write data outside the memory buffer which is not intended to store this data. When this happens we are talking about a buffer overflow or buffer overrun situation. A memory buffer is an area in the computer's memory (RAM) meant for temporarily storing data. This kind of buffers can be found in all programs and are used to store data for input, output and processing.

An example of data stored in buffers are login credentials or the hostname for an FTP server. Also other data temporarily stored before processing can be stored in buffers. This literally could be anything from user input fields such as username and password fields to input files used to import certain configuration files. When the amount of data written to the buffer exceeds the expected amount of data, the memory buffer is overrun. This happens for example when a username with a maximum of 8 bytes is expected and a username of 10 bytes is given and written to the buffer. In this case the buffer is exceeded by 2 bytes and an overflow will occur when it's not prevented from happening. This often happens due to bad programming and the lack of input sanitization.

TOP TUTORIALS

BY HACKING TUTORIALS – FEBRUARY 2, 2018 10

The Best Hacking Books 2018

BY HACKING TUTORIALS – NOVEMBER 15, 2016 2

Hacking with Netcat part 2: Bind and reverse shells

BY HACKING TUTORIALS – MAY 1, 2016 12

Metasploit commands

BY HACKING TUTORIALS – JULY 16, 2015 15

The Top 10 Wifi Hacking Tools in Kali Linux

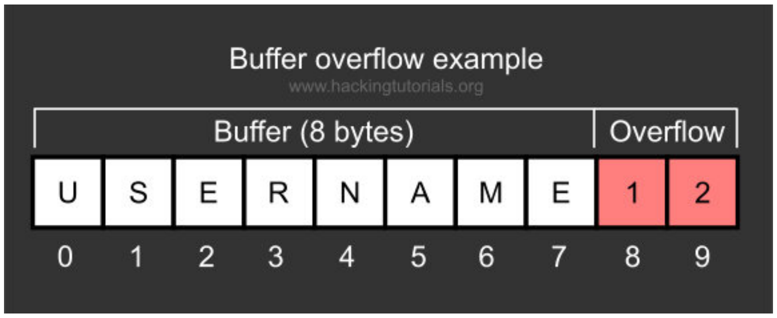
BY HACKING TUTORIALS – APRIL 18, 2018 0

Vulnerability Scanning with OpenVAS 9 part 1: Installation & Setup

BY HACKING TUTORIALS – JUNE 3, 2015 20

How to hack a WordPress website with WPScan

SUBSCRIBE



An example of a buffer overflow when writing 10 bytes of data (username12) to an 8 byte buffer.

What happens when a buffer overflow occurs?

When a memory buffer overflow occurs and data is written outside the buffer, the running program may become unstable, crash or return corrupt information. The overwritten parts of memory may have contained other important data for the running application which is now overwritten and not available to the program anymore. Buffer overflows can even run other (malicious) programs or commands and result in arbitrary code execution.

Arbitrary code execution and privilege escalation

When a buffer overflow vulnerability is used to write malicious data in the memory and the attacker is able to take control of the execution flow of a program, we are dealing with a serious security vulnerability. Buffer overflows can then become serious security issues. These security issues can be exploited by hackers to take (remote) control of a host, perform privilege escalation or a lot more bad things as a result of arbitrary code execution. Arbitrary code execution is the process of injecting code in the buffer and get it to execute.

Privilege escalation is performed through exploiting a buffer overflow vulnerability to execute arbitrary code in a program that is running with system privileges. The executed code can be shellcode which gives the attacker an OS shell with administrative privileges for example, or even add a new (administrator) user to the system. Also with buffer overflows the executed code happens in the context of the running application. This means that when the exploited application runs under with administrative privileges, the malicious code will also be executed with administrative privileges.

Denial of Service (DoS)

Not all buffer overflow vulnerabilities can be exploited to gain arbitrary code execution. Also (remote) Denial of Service attacks can be performed when they only crash the running program. As buffer overflows vulnerabilities can occur in any software DoS attacks are not just limited to services and computers. Also routers, firewalls IoT devices and anything else running an OS can be targeted. An example of this situation is the recent Cisco ASA IKEv1 and IKEv2 Buffer Overflow exploits lately. Some of these remote exploits only crash and force reboot the firewall resulting in a couple minutes downtime.

How to prevent buffer overflows from occurring?

Buffer overflows in software can be prevented or mitigated in several ways. Mitigation is the process of minimizing the impact of a threat before or after the threat occurs. This is exactly what we need to do when it comes to buffer overflows. They can be prevented from happening before they occur (proactive). But, since buffer overflows keep occurring, despite the proactively taken actions to avoid them, we also need mechanisms in place to minimize impact when they do occur (reactive countermeasures). Let's have a look at how buffer overflow prevention and mitigation works.

Buffer overflow prevention

The best and most effective solution is to prevent buffer overflow conditions from happening in the code. For example when a maximum of 8 bytes as input data is expected, than the amount of data which can be written to the buffer to be limited to 8 bytes at any time. Also, programmers should be using save functions, test code and fix bugs accordingly. Proactive methods for buffer overflow prevention like these should be used whenever possible to limit buffer overflow vulnerabilities.

Buffer overflow mitigation

Another way of safeguarding to buffer overflows is to detect them as they happen and mitigate the situation. This is an reactive approach and focuses on minimizing the harmful impact. An example of effective mitigation is a modern operating system which protects certain memory areas from being written to or executed from. This will prevent an attacker from writing arbitrary code to the memory when a buffer overflow occurred. Implementations like DEP, ASLR, SEHOP and executable space and pointer protection try to minimize the negative impact of a buffer overflow. This does not prevent the buffer overflow from occurring, but it does minimize the impact.

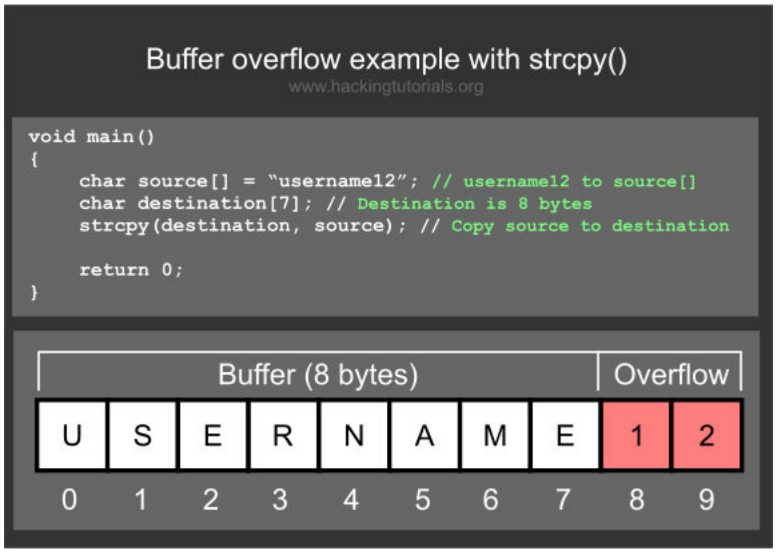
Another way of passive buffer overflow detection is using intrusion detection systems (IDS) to analyse network traffic. An IDS is capable of detecting signatures in network traffic which are known to exploit buffer overflow vulnerabilities. The IDS can than mitigate the attack and prevent the payload from executing on the targeted system.

How does a buffer overflow work and look like in code?

Let's have a look at how a buffer overflow actually works by looking at the program code. We explain this process using a very known function vulnerable to buffer overflow is the strcopy() function in the c library. This functions uses 2 pointers as parameters, the source which points to the source array to copy from and the destination pointer to the character array to write to. When the function is executed the source array of chars will be copied to the destination array and does not have a check for bounds when it does so. When the source buffer is larger than the destination buffer, than the buffer is overrun.

Buffer overflow with strcpy()

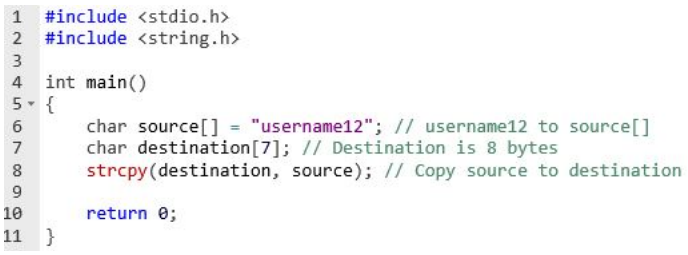
The follow image is an example of the strcpy() function using a source which is overrunning the destination buffer.



```
void main()
{
    char source[] = "username12"; // username12 to source[]
    char destination[7]; // Destination is 8 bytes
    strcpy(destination, source); // Copy source to destination

    return 0;
}
```

The code would look like the following image in you IDE of choice:



```
#include <stdio.h>
#include <string.h>

int main()
{
    char source[] = "username12"; // username12 to source[]
    char destination[7]; // Destination is 8 bytes
    strcpy(destination, source); // Copy source to destination

    return 0;
}
```

In this example the buffer is overrun with 2 bytes containing a harmless 1 and 2. Since the strcpy() function does not perform a bounds check we could write anything outside the buffer space. Also malicious code like shellcode. In the following tutorials about buffer overflows we will learn about overrunning buffers with shellcode instead of 1's and 2's. We will also learn how to control the execution flow of a program and execute the malicious shellcode outside the buffer.

Lessons learned

We have learned that a buffer overflow is caused by certain conditions where a running program is writing data outside the memory buffer. By injecting (shell)code and redirecting the execution flow of a running program to that code, an attacker is able to execute that code. This is called arbitrary code execution. With arbitrary code execution an attacker is able to gain (remote) control of a specific target, elevate privileges or cause a denial of service on the target.

Buffer overflows can be proactively prevented and mitigated with several techniques. Programmers should write secure code and test it for buffer overflows. When a buffer overflow is not prevented from happening it can still be mitigated with reactive methods like protecting memory from being written to.

We have tried to explain buffer overflow basics without to many technical details. In the following tutorials about this subject we will get into more details regarding stack based buffer overflows, heap based buffer overflows and how to detect and exploit buffer overflows vulnerabilities in software. We will also be learning about shellcode and writing our own basic buffer overflow exploits.

SHARE.

PREVIOUS ARTICLE NEXT ARTICLE

Hacking Tutorials 2017 | Upgrading Netcat shells to Meterpreter sessions

RELATED POSTS

BY HACKING TUTORIALS

- SEPTEMBER 14, 2017 3

Metasploitable 3: Exploiting HTTP PUT

BY HACKING TUTORIALS

- MAY 2, 2017 4

Eternalromance: Exploiting Windows Server 2003

BY HACKING TUTORIALS

- APRIL 18, 2017 38

Exploiting Eternalblue for shell with Empire & Msfconsole

3 COMMENTS

SWAGCAT on JUNE 3, 2017 9:56 PM

Thank you. Waiting next part of exploitation this vuln part of code!

REPLY

SIROJULMIFTAKH on JULY 8, 2017 7:03 AM

Is it possible that the vulnerability could occur in programming like php which does not need to be given the definition of data types on variables? *sorry for my english*

REPLY

HACKING TUTORIALS on JULY 8, 2017 8:51 AM

Yes that is very possible, have a look at the change logs related to buffer overflow and memory bugs: http://php.net/ChangeLog-5.php

REPLY

LEAVE A REPLY

Your Comment

Your Name

Your Email

Your Website

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.

POST COMMENT

RECENT TUTORIALS

- Vulnerability Scanning with OpenVAS 9 part 3: Scanning the Network
- Vulnerability Scanning with OpenVAS 9 part 2: Vulnerability Scanning
- Vulnerability Scanning with OpenVAS 9 part 1: Installation & Setup
- The Best Hacking Books 2018
- Hacking Tutorials 2018
- Enumerating LinkedIn with Inspy

POPULAR TUTORIALS

BY HACKING TUTORIALS – SEPTEMBER 1, 2016 113
Review: Offensive Security Certified Professional (OSCP)

BY HACKING TUTORIALS – APRIL 18, 2017 38
Exploiting Eternalblue for shell with Empire & Msfconsole

BY HACKING TUTORIALS – MARCH 17, 2016 36
Installing VPN on Kali Linux 2016 Rolling

FEATURED DOWNLOADS

- directory_scanner.py (15095 downloads)
- PEiD Userdb (9643 downloads)
- PEiD-0.95-20081103.zip (16485 downloads)
- wifi_jammer.py (21059 downloads)