Insecure Temporary File

Description

Creating and using insecure temporary files can leave application and system data vulnerable to attacks.

Applications require temporary files so frequently that many different mechanisms exist for creating them in the C Library and Windows® API. Most of these functions are vulnerable to various forms of attacks.

Risk Factors

TBD

Examples

The following code uses a temporary file for storing intermediate data gathered from the network before it is processed.

```
...
if (tmpnam_r(filename)){
    FILE* tmp = fopen(filename,"wb+");
    while((recv(sock,recvbuf,DATA_SIZE, 0) >
0)&&(amt!=0))
        amt = fwrite(recvbuf,1,DATA_SIZE,tmp);
}
...
```

The OWASP® Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Important Community Links

Community Attacks
Vulnerabilities (You
Controls

This otherwise unremarkable code is vulnerable to a number of different attacks because it relies on an insecure method for creating temporary files. The vulnerabilities introduced by this function and others are described in the following sections. The most egregious security problems related to temporary file creation have occurred on Unix-based operating systems, but Windows applications have parallel risks. This section includes a discussion of temporary file creation on both Unix and Windows systems.

Methods and behaviors can vary between systems, but the fundamental risks introduced by each are reasonably constant.

The functions designed to aid in the creation of temporary files can be broken into two groups based whether they simply provide a filename or actually open a new file.

Group 1 – "Unique" Filenames

The first group of C Library and WinAPI functions designed to help with the process of creating temporary files do so by generating a unique file name for a new temporary file, which the program is then supposed to open. This group includes C Library functions like tmpnam(), tempnam(), mktemp() and their C++ equivalents prefaced with an _ (underscore) as well as the GetTempFileName() function from the Windows API. This group of functions suffers from an underlying race condition on the filename chosen. Although the functions guarantee that the filename is unique at the time it is selected, there is no mechanism to prevent another process or an attacker from creating a file with the same name after it is selected but before the application attempts to open the file. Beyond the risk of a legitimate collision caused by another call to the same function, there is a high probability that an attacker will be able to create a collision because the filenames generated by these functions are not sufficiently randomized to make them difficult to guess.

If a file with the selected name is created, then depending on how the file is opened the existing contents or access permissions of the file may remain intact. If the existing contents of the file are malicious in nature, an attacker may be able to inject dangerous data into the application when it reads data back from the temporary file. If an attacker pre-creates the file with relaxed access permissions, then data stored in the temporary file by the application may be accessed, modified or corrupted by an attacker. On Unix based systems an even more insidious attack is possible if the attacker pre-creates the file as a link to another important file. Then, if the application truncates or writes data to the file, it may unwittingly perform damaging operations for the attacker. This is an especially serious threat if the program operates with elevated permissions.

Finally, in the best case the file will be opened with the a call to `open()` using the `O_CREAT` and `O_EXCL` flags or to `CreateFile()` using the `CREATE_NEW` attribute, which will fail if the file already exists and therefore prevent the types of attacks described above. However, if an attacker is able to accurately predict a sequence of temporary file names, then the application may be prevented from opening necessary temporary storage causing a denial of service (DoS) attack. This type of attack would not be difficult to mount given the small amount of randomness used in the selection of the filenames generated by these functions.

Group 2 – “Unique” Files

The second group of C Library functions attempts to resolve some of the security problems related to temporary files by not only generating a unique file name, but also opening the file. This group includes C Library functions like `tmpfile()` and its C++ equivalents prefaced with an _ (underscore), as well as the slightly better-behaved C Library function `mkstemp()`

The `tmpfile()` style functions construct a unique filename and open it in the same way that fopen() would if passed the flags `wb+`, that is, as a binary file in read/write mode. If the file already exists, `tmpfile()` will truncate it to size zero, possibly in an attempt to assuage the security concerns mentioned earlier regarding the race condition that exists between the selection of a supposedly unique filename and the subsequent opening of the selected file. However, this behavior clearly does not solve the function's security problems. First, an attacker can pre-create the file with relaxed access-permissions that will likely be retained by the file opened by tmpfile(). Furthermore, on Unix based systems if the attacker pre-creates the file as a link to another important file, the application may use its possibly elevated permissions to truncate that file, thereby doing damage on behalf of the attacker. Finally, if `tmpfile()` does create a new file, the access permissions applied to that file will vary from one operating system to another, which can leave application data vulnerable even if an attacker is unable to predict the filename to be used in advance.

Finally, `mkstemp()` is a reasonably safe way to create temporary files. It will attempt to create and open a unique file based on a filename template provided by the user combined with a series of randomly generated characters. If it is unable to create such a file, it will fail and return -1. On modern systems the file is opened using mode 0600, which means the file will be secure from tampering unless the user explicitly changes its access permissions. However, `mkstemp()` still suffers from the use of predictable file names and can leave an application vulnerable to denial of service attacks if an attacker causes `mkstemp()` to fail by predicting and pre-creating the filenames to be used.

Related Attacks

- Denial of Service

Related Vulnerabilities

Related Controls

Related Coding

- Python creating a temporary file the most secure manner possible

References

TBD

Edit on GitHub

Spotlight: Bionic

Bionic helps customers manage the security posture of their applications in production, providing continuous visibility of risk across all application services, dependencies, and data flows in real-time. Current application security tools are looking at data privacy and application security from a vulnerability lens. Bionic looks at the problem from an architectural lens.

Corporate Supporters

Become a corporate supporter