<UWE-484-03>: <Fixing Hardcoded Admin Cookie Key Vulnerability>

The Basics

Authors: Lanting Lu, Huijie Li
Disclosure Date: 03/16/2025
Product: tinyserv
Reporter(s): CSE 484 staff
Exploit sample: see sploit3.sh

The Exploit

Exploit strategy (2p) - What does the exploit do? What are the key parts of the exploit?:
The exploit script aims to access a protected admin.txt file sending a crafted HTTP GET request. It utilizes a specially generated cookie to bypass normal authentication checks, potentially exploiting weaknesses in how the server validates cookies.
The key parts of the exploit is that since tinyserv is is an open source, so the attacker is able to see how the cookie is generated and the hard coded admin_cookie_key. So he may be able to generate a cookie via sploit3_cookiegen to generate a fake cookie that might be used to trick the server into thinking the request comes from an authenticated session. Using the extracted key from the original code, the attacker locally run a modified version of the make_cookie function or an equivalent script to generate valid but unauthenticated cookies.

Exploit primitives (2p) - What new capabilities (if any) does an attacker gain from successful exploitation of the vulnerability? These are small, process-space or environment-related building blocks used to construct full exploit chains such as: write arbitrary data to arbitrary memory locations, overwrite local variables, overwrite heap data, crash the program, send data on the network, read from local files, escalate privileges, etc.:
If the vulnerability involving the hardcoded cookie key is successfully exploited, the attacker can forge authentication cookies, which would enable unauthorized access to protected system areas and data manipulation. Also, the attacker can tamper with data stored on the heap, potentially overwriting data or variables to modify the application's behavior, such as reading from local files, or even launching network attacks from the compromised system.

The Vulnerability

Vulnerability details (3p) - Provide details about the vulnerability. Also, if applicable, provide the vulnerable file(s), function(s), and line number(s):
The vulnerability depends on a hard coded admin_cookie, which is used to generate a cookie for authentication by hmac_sha1.
We notice that in line 46-51, it states: const uint8_t *admin_cookie_key, which is a globally defined cookie key in the source file. The hard-coded cookie key is accessible to anyone who can

view the source code. This allows attackers to extract the key and use it to forge authentication cookies. Such a vulnerability enables attackers to bypass authentication mechanisms and gain unauthorized access to protected areas of the application.
In line 1018:

```
 hmac_sha1(admin_cookie_key, sizeof(admin_cookie_key),
                    (uint8_t *) to_sign, sizeof(cookie_data), (uint8_t *) signature);
   // write cookie data
   int data_size = sizeof(cookie_data);
```

Bug class (CWE) (1p) - Visit this page (https://cwe.mitre.org/data/definitions/699.html) and provide the name and CWE identifier number of the bug class that best fits the vulnerability. You will need to expand the dropdowns/bullets to see all the options for each category.:

CWE-798: Use of Hard-coded Credentials

Severity/priority score (2p) - Rate the severity of the vulnerability as one of low, medium, or high, and explain your choice in terms of the vulnerability and the provided exploit:

Medium

This vulnerability exists due to a hardcoded admin_cookie_key in the source code of tinyserv. Since tinyserv is open-source, an attacker can easily retrieve the key by inspecting the source code. Once the attacker obtains the key, they can forge authentication cookies to gain unauthorized access to admin.txt, which contains sensitive logs of login attempts.

The issue falls under CWE-798 (Use of Hard-coded Credentials), as it involves a hardcoded authentication key stored in the source code, making it easily accessible to attackers who can use it to bypass authentication and gain unauthorized access.

Since admin.txt stores login attempts, if an admin enters their correct credentials, this file may contain sensitive information, including valid usernames and passwords. A successful attacker could retrieve this file, leading to information disclosure and potential further system compromise.

Thoughts on how this vulnerability might have been found by the attacker (1p) - What did the attacker do (what tools, techniques, etc.) to find this vulnerability?

The attacker could have found this vulnerability through manual inspection, where they examine the source code for hard-coded cryptographic keys. Since tinyserv is open-source, an attacker could easily access the source code and search for sensitive information such as predefined constants or variables that store key values. Additionally, attackers might use automated tools like static analysis tools or grep commands to quickly locate hard-coded credentials.

The Patch

Proposed patch plan (2p) - This must be something you can implement in code for tinyserv; do not say something like "use stack canaries" for your answer to this question:

The patch plan is to replace the hardcoded admin_cookie_key with a dynamically generated key that is securely stored in an external file (cookie_key.txt). We will implement a function to generate a new random admin_cookie_key when the server starts and store it in cookie_key.txt. Instead of hardcoding the key in the source code, the server will read the key from the file when authentication is needed.
We will use the getrandom function to generate 32-byte secure random values and encode them in hexadecimal format. The generated key will be stored in the admin_cookie_key field and null-terminated (admin_cookie_key[key_length * 2] = '\0'). The server will check if cookie_key.txt exists before generating a new key. If the file is missing, a new key will be generated and stored securely. To prevent unauthorized access, strict file permissions will be enforced so that only the server process can read the file.
When an admin logs in, the server will read cookie_key.txt to retrieve the key and validate authentication. This approach eliminates the risk of key leakage from source code while allowing for easier key rotation. However, if cookie_key.txt is lost or corrupted, authentication will fail, so a backup strategy must be in place.

Killing the bug class (bonus 1p) - What could the tinyserv project do to prevent more vulnerabilities of this CWE type from appearing anywhere in this codebase in the future? Explain some advantages and disadvantages of your solution:

To address CWE-798 (Hard-coded Credentials), the Tinyserv project should implement an external encrypted configuration file system to store all cryptographic keys, which would be decrypted and loaded at runtime with proper access controls limiting file permissions to authorized processes only. This approach offers several advantages: it prevents key exposure if source code is compromised, enables straightforward key rotation without code modifications, centralizes credential management, and aligns with security best practices. However, this solution comes with trade-offs: it introduces additional deployment complexity, creates potential points of failure if the key file becomes corrupted or inaccessible, requires careful permission management across environments, and necessitates robust error handling for file-related issues. While this approach can't completely prevent developers from introducing new hardcoded credentials, it establishes a clear pattern and infrastructure that makes secure credential management the path of least resistance, ultimately helping to prevent similar vulnerabilities throughout the codebase.