SUBGRAPH

DRAFT FINDINGS REPORT

FileZilla

April 4, 2022

Prepared for: FileZilla

Subgraph Technologies, Inc.
642 Rue de Courcelle, Suite 309
Montreal, Quebec
https://subgraph.com

Contents

Overview

Subgraph conducted a security audit of the FileZilla FTP Server on behalf of the Open Technology Fund's Red Team Lab program. The audit was performed in Q1 2022. This work was in support of the Open Technology Fund's Red Team Labs initiative to provide professional security audits to projects that advance Internet Freedom goals.

The objective of this engagement was to comprehensively review the FileZilla FTP server, which is an open source project developed in C++ that implements the file transfer protocol (FTP).

Scope

The scope included the following:

- Security testing of general functional components, including the basic protocol implementation
- Evaluating the impersonator component which is used for enabling OS users to authenticate as themselves, as opposed to the FileZilla FTP users traditionally supported
- Support for TLS
- Automated security testing for implementation errors, such as memory corruption
- Review of relevant components of dependencies, including libfilezilla, which is shared with the FileZilla client

The scope excluded the following:

- FileZilla client
- Denial of service mitigations

Subgraph performed the engagement with three different approaches:

1. Deploying the FTP server and manually performing simulated attack experiments with a methodology informed by threat modeling
2. Tactical code review informed by threat modeling and manual testing of the deployed server
3. Automated fuzz testing

High Level Methodology

For the FTP session testing, Subgraph simulated malicious FTP clients. The authentication levels included:

- No authentication (pre-auth)
- FileZilla users
- OS users and the impersonator privilege separation design
- Administrators accessing through the administrative service

Subgraph also tested scenarios where MITM interception is assumed possible and relevant to threat model, i.e., for the HTTPS client dispatched to execute the ACME protocol for Let's Encrypt enrollment.

Additionally, simulated testing scenarios included circumstances where there is a cooperating local user with system access, and circumstances where there is not such a user.

Specific interactive/manual testing methodology was derived from examination of supported FTP protocol commands and extensions, and from the relationship between the FileZilla Server and the host OS where it can be run.

Code review was performed tactically, driven by identification of threat boundaries, observed behavior, and known characteristics of underlying OS platforms.

Fuzz testing was very naive, entirely automated, and targeted the authenticated and unauthenticated attack surface of the FTP server as it is exposed to remote clients with the minimum FTP vocabulary.

Testing Environment

FileZilla Server version 1.2.0, and 1.1.0 built from source running on Linux, using Ubuntu and Alpine (in a Docker runtime) for the host OS.

FileZilla Server 1.2.0 was built against libfilezilla-0.35, GnuTLS 3.7.1-r0, nettle 3.7.3-r0, pugixml 1.11.4-r1, and GNU libstdc++-10.3.1.

Binary distributions of FileZilla Server 1.2.0 and 1.3.0 running on Windows 11 professional.

The configurations were as default except for a custom *users.xml* in which we authorized system user logins and define two FileZilla users. The host filesystem state was arbitrary and adjusted during testing to cover various circumstances and scenarios.

Observations

Authentication

FileZilla Server supports two types of users who can login: FileZilla users, configured in the *users.xml* configuration file, and users from the host OS, who login using their local credentials. The latter are able to login if the server is configured to permit this, which is not a default configuration in the versions tested.

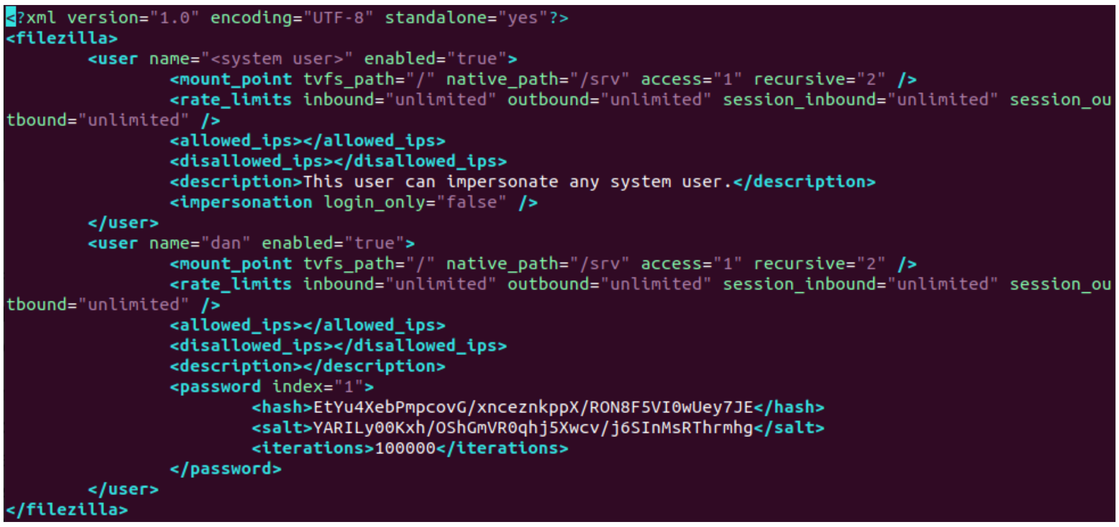
Host authentication is performed by the *libfilezilla check_auth()* function in *lib/impersonation.cpp*. Users on Linux servers are authenticated by comparing a hash of the supplied password to the hash in the host */etc/shadow* file. Windows users are authenticated using the the supplied username and password as parameters to the LogonUserW() function provided by the Win32 system API.

Following authentication of OS users, FileZilla Server eventually creates an *impersonator* object, further described in its own subsection, which exists for the duration of the authenticated session. The *impersonator* object is used for enforcing a lower privilege bound on requests to interact with securable objects such as files or directories, along with a lower-privileged process which executes filesystem operations. The *impersonator* abstraction wraps the identity and entitlements specific to the underlying OS.

FileZilla Server Passwords

Server user passwords are hashed before being persisted. There are three choices for digest algorithms: MD5, PBKDF2+SHA256, and SHA512. MD5 and SHA512 are documented as being present for backwards compatibility and are not recommended for production use. The default choice is PBKDF2+SHA256.

The password hash is computed from a SHA256 digest of the password data and a 32-byte salt. The output of this value after a minimum of 100000 iterations of PBKDF2 is serialized with the salt value and the number of PBKDF2 iterations. The implementation of this is largely provided by the *Nettle* function *nettle_pbkdf2_hmac_sha256()*. The serialized credential and metadata are stored in *users.xml*.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<filezilla>
        <user name="<system user>" enabled="true">
                <mount_point tvfs_path="/" native_path="/srv" access="1" recursive="2" />
                <rate_limits inbound="unlimited" outbound="unlimited" session_inbound="unlimited" session_ou
tbound="unlimited" />
                <allowed_ips></allowed_ips>
                <disallowed_ips></disallowed_ips>
                <description>This user can impersonate any system user.</description>
                <impersonation login_only="false" />
        </user>
        <user name="dan" enabled="true">
                <mount_point tvfs_path="/" native_path="/srv" access="1" recursive="2" />
                <rate_limits inbound="unlimited" outbound="unlimited" session_inbound="unlimited" session_ou
tbound="unlimited" />
                <allowed_ips></allowed_ips>
                <disallowed_ips></disallowed_ips>
                <description></description>
                <password index="1">
                        <hash>EtYu4XebPmpcovG/xnceznkppX/RON8F5VI0wUey7JE</hash>
                        <salt>YARILy00Kxh/OShGmVR0qhj5Xwcv/j6SInMsRThrmhg</salt>
                        <iterations>100000</iterations>
                </password>
        </user>
</filezilla>
```

Random Number Generation

FileZilla Server relies on randomness for salt, nonce, and key generation. The underlying source for random bytes varies: on Windows, CryptGenRandom() is used, while Linux servers will source using getrandom() or getentropy() or by reading from */dev/urandom*, depending on availability.

Impersonator

FileZilla Server relies on a component called the *impersonator* to implement support for system users and filesystem access controls rather than the FileZilla users and permissions defined in the server configuration.

The *impersonator* process is a separate executable; it is an agent that performs filesystem I/O operations while running with privileges of system users who have authenticated FTP sessions with FileZilla. FileZilla Server itself maintains elevated privileges and uses file handles (file descriptors, in the case of Linux, an anonymous pipe on Windows) created by the lower privileged *impersonator process*, to rely on OS and filesystem access control. Duplication of file handles on Windows and using **SCM_RIGHTS** to pass file descriptors on Linux are the OS facilities that are relied upon to delegate I/O across the trust boundary in FileZilla Server's IPC architecture.

Authorization

FileZilla Server implements an overlay filesystem abstraction that conforms to TVFS when exposed to FTP clients. The TVFS implementation maintains a filesystem tree independently of the actual filesystem state, updating mappings through client initiated requests and operations. There are authorization checks that are applied to this model, which presented possible opportunities for race conditions and vulnerabilities related to state confusion and disagreement between the FileZilla filesystem tree and the host filesystem, however in practice the *impersonator* was effective by design in mitigating attempts to craft practical attacks. This does not mean that such attacks do not exist, but we could not identify any instances within the testing period.

Administration Service

There is an administration service which is used with the client to configure aspects of the service while it is running. The server for this listens on loopback interfaces.

Support for TLS

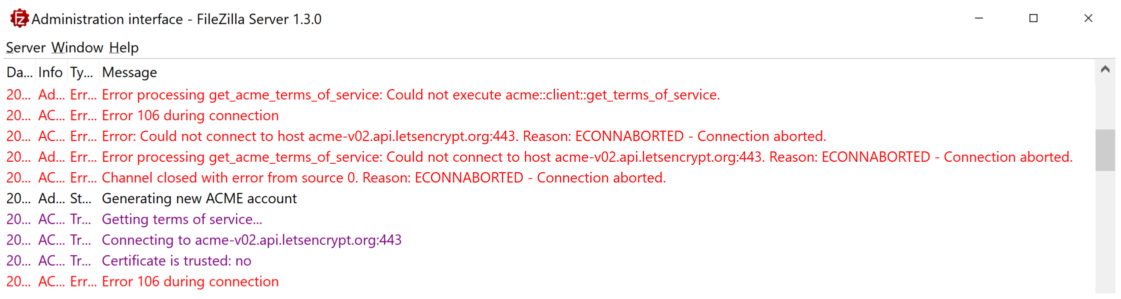
FileZilla Server uses the TLS layer implementation in libfilezilla. The libfilezilla TLS layer relies on GnuTLS and Nettle for lower level primitives, including the cryptographic elements. The TLS layer in libfilezilla is flexible, permitting, for example, certificate validation using the system trust store or more relaxed Trust on First Use (*TOFU*) model presumably intended for the FileZilla client.
The FileZilla client and server share the same TLS validation logic in the libfilezilla TLS layer, an observation

relevant with respect to possible behavior of the ACME HTTPS client that is used for Let's Encrypt enrollment in the server. This was tested and is noted as part of the following observation.

Let's Encrypt

FileZilla supports the self-service procurement of free domain-verified CA-signed certificates through a GUI-driven process that can be initiated from the administration interface. The mechanism dispatches an HTTPS client in the background that enrolls and completes the registration and challenge process. This involves client-side generation of a secp256r1 EC keypair which is used to sign a nonce sent by the Let's Encrypt directory server. The implementation of this uses Nettle called from within libfilezilla.

Subgraph tested the ACME HTTPS client to observe behavior when TLS certificate validation fails against the system trust store. The Let's Encrypt enrollment process was observed to close failed in this case, with a trust failure message output if verbose debug output is enabled.



```
Da... Info Ty...  Message
20...  Ad... Err... Error processing get_acme_terms_of_service: Could not execute acme::client::get_terms_of_service.
20...  AC... Err... Error 106 during connection
20...  AC... Err... Error: Could not connect to host acme-v02.api.letsencrypt.org:443. Reason: ECONNABORTED - Connection aborted.
20...  Ad... Err... Error processing get_acme_terms_of_service: Could not connect to host acme-v02.api.letsencrypt.org:443. Reason: ECONNABORTED - Connection aborted.
20...  AC... Err... Channel closed with error from source 0. Reason: ECONNABORTED - Connection aborted.
20...  Ad... St...  Generating new ACME account
20...  AC... Tr...  Getting terms of service...
20...  AC... Tr...  Connecting to acme-v02.api.letsencrypt.org:443
20...  AC... Tr...  Certificate is trusted: no
20...  AC... Err... Error 106 during connection
```

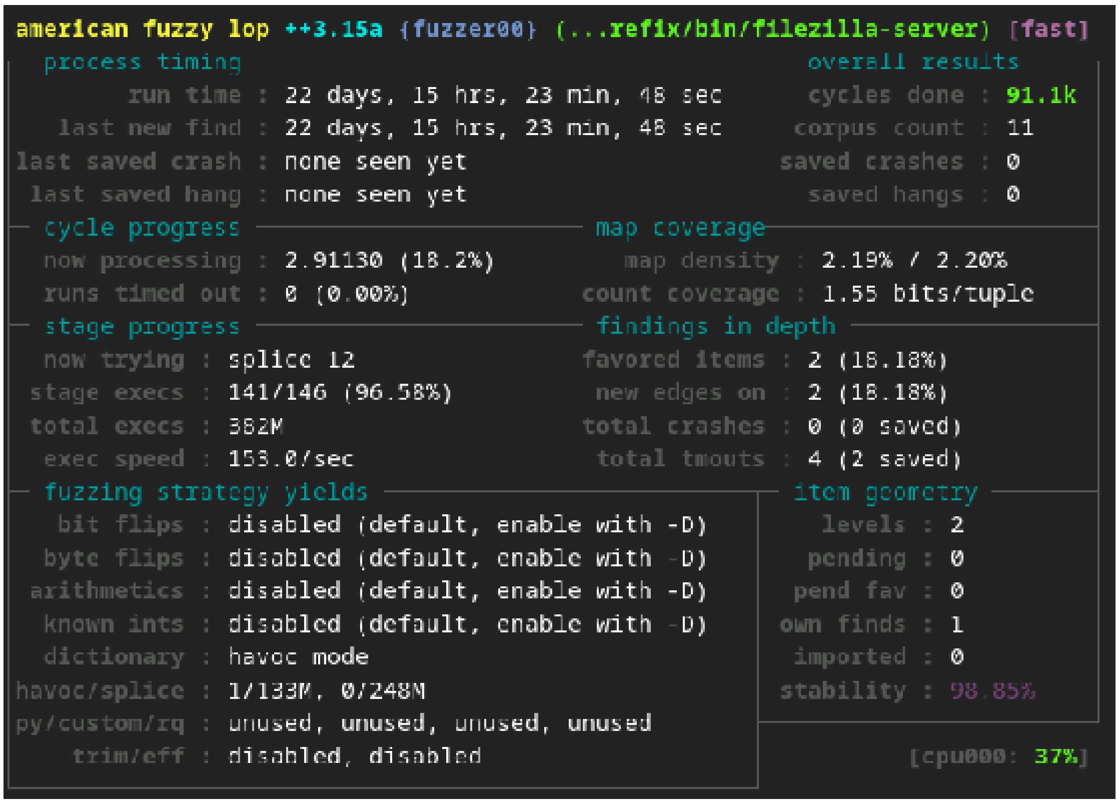
Self-Signed Certificates

FileZilla generates self-signed certificates for use when there is no alternative certificate set. The implementation that performs this is in the libfilezilla TLS layer. The certificates that are generated use ECDSA with the **GNUTLS_SEC_PARAM_HIGH** (128-bit) option set for EC key generation. The libfilezilla code appears to have deprecated RSA, though implementation details for RSA key generation remain present in the source unit. When RSA was last supported it specified a default modulus size of 2048 bits.

The validity period of self-signed certificates is approximately 366 days from the system time of generation.

Memory Corruption, Race Conditions, and General Code Observations

libfilezilla and FileZilla Server are written in C++ using a contemporary syntactic style and modern C++ features. With respect to avoidance of memory corruption, there appeared to be careful awareness and management of types. The use of integer types to manage indices and bounds checking were also reviewed in places in an attempt to find instances where over/underflows and signedness confusion could introduce vulnerabilities; none were located, though our review at this level of detail was not comprehensive and informed by threat modeling and intuition. Brute force was also applied with fuzz testing using AFL++ and no crashes were observed during a period of testing that lasted approximately a month.



```
american fuzzy lop ++3.15a {fuzzer00} (...refix/bin/filezilla-server) [fast]
│  process timing                                       overall results  │
│        run time : 22 days, 15 hrs, 23 min, 48 sec      cycles done : 91.1k │
│   last new find : 22 days, 15 hrs, 23 min, 48 sec     corpus count : 11   │
│last saved crash : none seen yet                     saved crashes : 0    │
│ last saved hang : none seen yet                       saved hangs : 0    │
├─ cycle progress ──────────────────────── map coverage─────────────────────┤
│  now processing : 2.91130 (18.2%)          map density : 2.19% / 2.20%   │
│  runs timed out : 0 (0.00%)             count coverage : 1.55 bits/tuple │
├─ stage progress ──────────────────────── findings in depth ───────────────┤
│  now trying : splice 12                 favored items : 2 (18.18%)       │
│ stage execs : 141/146 (96.58%)           new edges on : 2 (18.18%)       │
│ total execs : 382M                      total crashes : 0 (0 saved)      │
│  exec speed : 153.0/sec                  total tmouts : 4 (2 saved)      │
├─ fuzzing strategy yields ──────────────────────────── item geometry ──────┤
│   bit flips : disabled (default, enable with -D)     levels : 2          │
│  byte flips : disabled (default, enable with -D)    pending : 0          │
│ arithmetics : disabled (default, enable with -D)   pend fav : 0          │
│  known ints : disabled (default, enable with -D)  own finds : 1          │
│  dictionary : havoc mode                           imported : 0          │
│havoc/splice : 1/133M, 0/248M                      stability : 98.85%     │
│py/custom/rq : unused, unused, unused, unused                             │
│    trim/eff : disabled, disabled                            [cpu000: 37%]
```

Subgraph also attempted to locate race conditions related to the filesystem operations but determined that the impersonator was fairly effective by design. There were instances where there appeared to be some disagreement between FileZilla Server's model of the filesystem and the host filesystem, but this was not practically useful when attacks were attempted due to the privilege separation. The server could be manipulated into misrepresenting the state of the filesystem, but attempts to access restricted resources was found to be prevented.

Summary

No.	Title	Severity	Remediation
V-001	Linux Impersonator Read Blocking Denial of Service Vulnerability	Low	Resolved
V-002	Log Forgery via File Descriptor Leakage	Low	Resolved
V-003	Limited CA Blocklist	Low	Unresolved
V-004	FileZilla Exposure to User Created Hardlinks	Low	Resolved

Details

V-001: Linux Impersonator Read Blocking Denial of Service Vulnerability

Severity	Remediation
Low	Resolved

Discussion

An impersonator process is spawned once a system user logs in and performs a filesystem operation that requires it, e.g. by issuing an CWD FTP command. The FileZilla server creates a channel for communication between itself and the impersonator, maintaining it for the life of the user session. On Linux this is an anonymous unix domain socket that is created with the *socketpair(2)* system call. This channel is used to communicate messages bidirectionally and pass file descriptors for I/O operations from the impersonator to FileZilla server. Filezilla abstracts this by wrapping it in an impersonator object associated with the session, which is an argument passed to FileZilla server methods that handle commands invoked by the FTP client.

In the following example, the impersonator is PID 3181 in the process list, running as an unprivileged user:

```
2ab82b20f499:~$ ps
PID   USER     TIME  COMMAND
 3181 hoho      0:00 /usr/local/bin/filezilla-server-impersonator
 MAGIC_VALUE! 14 14
```

If *kernel.yama.ptrace_scope* is set to 0, local user *hoho* can use *ptrace(2)* to attach to the impersonator process associated with their FTP session.

The impersonator can cause the FileZilla Server to block indefinitely, resulting in a denial of service for all clients. The following can be performed to demonstrate the issue:

Step 1: Write a single byte of value *0x1* to the IPC file descriptor will cause the server to block. A simple way to do this is something like this (assume channel fd 14 and pid 3181):

```
2ab82b20f499:~$ echo 'call (size_t)write(14,"\x1",1)' | gdb -p 3181
```

Step 2: Issue a command in the FTP client that will send a message to the impersonator, relying on a callback function to handle the response. The CWD command is used in this example, but others work as well:

```
2022-01-21T14:42:08.024Z >> [FTP Session 2 127.0.0.1 hoho] CWD filezilla-
server-1.2.0
```

The server then blocks indefinitely because the callback function will read the byte from the channel as a non-error message response, and does not time out attempting to read more. The generic code that implements this for all message response callbacks is in *filezilla-server-1.2.0/src/filezilla/impersonator/channel.hpp*:

```
[..]
 if (!error_) {
                caller_.logger_.log(logmsg::error, L"[%s]: waiting
        for message", util::type_name<T>());

                any_message any;

                error_ = caller_.channel_.recv(any);
                if (error_) {
                        caller_.logger_.log(logmsg::error, L"[%s]: could
            not read response from the server: %s (%d)",
            util::type_name<T>(), std::strerror(error_), error_);
                        return false;
                }
[..]
```

The error check will pass if a byte with value *0x1* is read from the socket and there is no further data to read.

Impact Analysis

The result is a block on *recv()* for the channel, completely disabling the FTP server for all users until it is manually restarted.

Remediation Recommendations

Subgraph reported this issue to the FileZilla Server team prior to the creation of this report. This was addressed with multiple improvements to make the impersonator more resistant to tracing in version 1.3.0. From the changelog:

- Linux: Warn if sysctl knob kernel.yama.ptrace_scope is 0

An additional stopgap measure that has been implemented for Linux is to use *prctrl(2)* with **PR_SET_DUMPABLE** to set **SUID_DUMP_DISABLE** for the impersonator process. Processes that are marked non-dumpable cannot be accessed using **PTRACE_ATTACH**. This was implemented with acknowledgement of the race condition present as an attacker can attach before this is done.

Additional Information

N/A

V-002: Log Forgery via File Descriptor Leakage

Severity	Remediation
Low	Resolved

Discussion

The user can use a similar method to the attack described in **V-001** to write to other file descriptors owned by the impersonator process. These file descriptors are inherited from the parent process, and the *stderr* stream can be used to write arbitrary data to the server log output stream that is not distinguished or identified as originating from the impersonator child process (i.e., from an unprivileged user).

Impact Analysis

This can be used to forge log messages. For example:

```
2022-01-10T02:18:32.123Z !! [Administration Server] User hoho uploads phreshest warez
2022-01-10T02:21:32.123Z !! [Administration Server] User haha is leech that
uploads shareware
```

Remediation Recommendations

This can be mitigated by closing or redirecting impersonator output, or, if it is desired to be kept, clearly marking it as such.

Note: This has been addressed in libfilezilla 0.36.0 with an io_redirect mode that can close parent-side handles for stdin/stdout/stderr are closed.

It may be worthwhile to read the sysctl value for *kernel.yama.ptrace_scope* and warn on the risk, or fail to start without a special command swich or configuration option to bypass.

Note: this has been implemented in FileZilla Server 1.3.0.

Additional Information

N/A

V-003: Limited CA Blocklist

Severity	Remediation
Low	Unresolved

Discussion

As part of its custom certificate validation logic, libfilezilla includes a check to determine if a certificate is blacklisted, due to some known compromise or malicious behavior. This check is performed by comparing the certificate KeyID against a list populated with only a single entry. See *tls_layer_impl::certificate_is_blacklisted()* in *lib/tls_layer_impl.cpp*:

```
[..]
bool tls_layer_impl::certificate_is_blacklisted(gnutls_x509_crt_t const& cert)
{
        static std::set<std::string, std::less<>> const
    bad_authority_key_ids = {
    std::string("\xF4\x94\xBF\xDE\x50\xB6\xDB\x6B\x24\x3D\x9E\xF7\xBE\x3A
        \xAE\x36\xD7\xFB\x0E\x05", 20) // Nation-wide MITM in Kazakhstan
        };
[..]
```

This KeyID corresponds to a specific certificate that was used in an attempt to establish a nation-wide MITM capability through local installation by users, something encouraged through a variety of means. Since then there have been other certificates known to be associated with other similar attempts.

Impact Analysis

Maintaining a list of known compromised or untrustworthy KeyIDs is not something most TLS implementations do. Browsers and some other software projects do it, but they do this independently and perhaps inconsistently from one another. In addition to this there are many other ways that local trust stores and the global PKI are undermined. This is in particular an issue for desktop systems.

However, because the feature exists in libfilezilla and FileZilla Server, and because it does contain a KeyID in its hardcoded list, there is arguably an expectation that it perform as designed. For this reason Subgraph has identified this as a finding.

Remediation Recommendations

Consider including some or all of the certificate and public keys listed in the Chromium blocklist that are associated with this attack, as well as some or all of the others:

Chromium blocklist

Note that the above link may change if the repository is restructured.

The FileZilla Server pointed out other threats that undermine trust in the system trust store and proposed a strategic solution of using a trust store that is separate and intended to be used by software such as the FileZilla client. This trust store could be installed with with the software and maintained independently. One such option is the Common CA Database.

That may make sense to integrate, epsecially for the client, if this is an attack that is to be included in its threat model.

Additional Information

Protecting Chrome users in Kazakhstan

Mozilla takes action to protect users in Kazakhstan

Censored Planet: Kazakhstan's HTTPS Interception

V-004: FileZilla Exposure to User Created Hardlinks

Severity	Remediation
Low	Resolved

Discussion

FileZilla Server on some Linux or other UNIX-like systems may be exposed to maliciously created hardlinks if a local user can write to an area of the filesystem exposed through a **FileZilla user** account that they also have access to. In the case of modern Linux, this attack would not be common, as the *fs.protected_hardlinks* sysctl switch prevents creation of hardlinks to files that unprivileged users do not own. However, on some older systems, or systems with custom kernels, *fs.protected_hardlinks* may not be enabled. There is a risk of privilege escalation through FileZilla Server if this protection does not exist or is not enabled.

Impact Analysis

A user who can write to a directory can simply create a hardlink to e.g. */etc/shadow* and then use FileZilla Server to retrieve it if that directory is exposed through FileZilla, as a FileZilla user. This is because FileZilla does not lower privileges for sessions with FileZilla users.

Remediation Recommendations

It may be worthwhile to read the sysctl value and warn on the risk, or fail to start without a special command switch or configuration option to bypass.

Note: This has been implemented in FileZilla 1.3.0.

Additional Information

[PATCH] fs: hardlink creation restrictions

Appendix

Methodology

Our approach to testing is designed to understand the design, behavior, and security considerations of the assets being tested. This helps us to achieve the best coverage over the duration of the test.

To accomplish this, Subgraph employs automated, manual and custom testing methods. We conduct our automated tests using the industry standard security tools. This may include using multiple tools to test for the same types of issues. We perform manual tests in cases where the automated tools are not adequate or reveal behavior that must be tested manually. Where required, we also develop custom tools to perform tests or reproduce test findings.

The goals of our testing methodology are to:

- Understand the expected behavior and business logic of the assets being tested
- Map out the attack surface
- Understand how authentication, authorization, and other security controls are implemented
- Test for flaws in the security controls based on our understanding
- Test every point of input against a large number of variables and observe the resulting behavior
- Reproduce and re-test findings
- Gather enough supporting information about findings to enable us to classify, report, and suggest remediations

Description of testing activities

Depending on the type and scope of the engagement, our methodology may include any of the following testing activities:

1. **Information Gathering:** Information will be gathered from publicly available sources to help increase the success of attacks or discover new vulnerabilities
2. **Network discovery:** The networks in scope will be scanned for active, reachable hosts that could be vulnerable to compromise
3. **Host Vulnerability Assessment:** Hosts applications and services will be assessed for known or possible vulnerabilities
4. **Application Exploration:** The application will be explored using manual and automated methods to better understand the attack surface and expected behavior
5. **Session Management:** Session management in web applications will be tested for security flaws that may allow unauthorized access
6. **Authentication System Review:** The authentication system will be reviewed to determine if it can be bypassed
7. **Privilege Escalation:** Privilege escalation checks will be performed to determine if it is possible for an authenticated user to gain access to the privileges assigned to another role or administrator

8. **Input Validation:** Input validation tests will be performed on all endpoints and fields within scope, including tests for injection vulnerabilities (SQL injection, cross-site scripting, command injection, etc.)
9. **Business Logic Review:** Business logic will be reviewed, including attempts to subvert the intended design to cause unexpected behavior or bypass security controls

Reporting

Findings reports are peer-reviewed within Subgraph to produce the highest quality findings. The report includes an itemized list of findings, classified by their severity and remediation status.

Severity ratings

Severity ratings are a metric to help organizations prioritize security findings. The severity ratings we provide are simple by design so that at a high-level they can be understood by different audiences. In lieu of a complex rating system, we quantify the various factors and considerations in the body of the security findings. For example, if there are mitigating factors that would reduce the severity of a vulnerability, the finding will include a description of those mitigations and our reasoning for adjusting the rating.

At an organization's request, we will also provide third-party ratings and classifications. For example, we can analyze the findings to produce *Common Vulnerability Scoring System* (CVSS)[1] scores or *OWASP Top 10*[2] classifications.

The following is a list of the severity ratings we use with some example impacts:

Critical

Exploitation could compromise hosts or highly sensitive information

Critical Exploitation could compromise hosts or highly sensitive information

High

Exploitation could compromise the application or moderately sensitive information

High Exploitation could compromise the application or moderately sensitive information

Medium

Exploitation compromises multiple security properties (confidentiality, integrity, or availability)

Medium Exploitation compromises multiple security properties (confidentiality, integrity, or availability)

[1] https://www.first.org/cvss/
[2] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Low

Exploitation compromises a single security property (confidentiality, integrity, or availability)

Low Exploitation compromises a single security property (confidentiality, integrity, or availability)

Info

Finding does not directly pose a security risk but merits further investigation

Info Finding does not directly pose a security risk but merits further investigation

The severity of a finding is often a product of the impact to general security properties of an application, host, network, or other information system.

The properties that can be impacted are:

Confidentiality Exploitation results in authorized access to data

Integrity Exploitation results in the unauthorized modification of data or state

Availability Exploitation results in a degradation of performance or an inability to access resources

The actual severity of a finding may be higher or lower depending on a number of other factors that may mitigate or exacerbate it. These include the context of the finding in relation to the organization as well as the likelihood of exploitation. These are described in further detail below.

Contextual factors

Confidentiality, integrity, and availability are one dimension of the potential risk of a security finding. In some cases, we must also consider contextual factors that are unique to the organization and the assets tested.

The following is a list of those factors:

Financial Exploitation may result in financial losses

Reputation Exploitation may result in damage to the reputation of the organization

Regulatory Exploitation may expose the organization to regulatory liability (e.g. make them non-compliant)

Organizational Exploitation may disrupt the operations of the organization

Likelihood

Likelihood measures how probable it is that an attacker exploit a finding.

This is determined by numerous factors, the most influential of which are listed below:

Authentication Whether or not the attack must be authenticated

Privileges Whether or not an authenticated attacker requires special privileges

Public exploit Whether or not exploit code is publicly available

Public knowledge Whether or not the finding is publicly known

Exploit complexity How complex it is for a skilled attacker to exploit the finding

Local vs. remote Whether or not the finding is exposed to the network

Accessibility Whether or not the affected asset is exposed on the public Internet

Discoverability How easy it is for the finding to be discovered by an attacker

Dependencies Whether or not exploitation is dependant on other findings such as information leaks

Remediation status

As part of our reporting, remediation recommendations are provided to the client. To help track the issues, we also provide a remediation status rating in the findings report.

In some cases, the organization may be confident to remediate the issue and test it internally. In other cases, Subgraph works with the organization to re-test the findings, resulting in a subsequent report reflecting remediation status updates.

If requested to re-test findings, we determine the remediation status based on our ability to reproduce the finding. This is based on our understanding of the finding and our awareness of potential variants at that time. To reproduce the results, the re-test environment should be as close to the original test environment as possible.

Security findings are often due to unexpected or unanticipated behavior that is not always understood by the testers or the developers. Therefore, it is possible that a finding or variations of the finding may still be present even if it is not reproducible during a re-test. While we will do our best to work with the organization to avoid this, it is still possible.

The findings report includes the following remediation status information:

Resolved
Finding is believed to be remediated, we can no longer reproduce it

Resolved Finding is believed to be remediation, we can no longer reproduce it

In progress
Finding is in the process of being remediated

In progress Finding is in the process of being remediated

Unresolved
Finding is unresolved – used in initial report or when the organization chooses not to resolve

Unresolved Finding is unresolved – used in initial report or when the organization chooses not to resolve

Not applicable
There is nothing to resolve, this may be the case with informational findings