# Anmeldung am FRITZ!Box Webinterface

## FRITZ!Box Schnittstellen

FRITZ!OS bietet mehrere Möglichkeiten, auf die Konfiguration des FRITZ!-Gerätes oder auf Statistiken und weitere Live-Daten zuzugreifen.

- Das TR-064 Protokoll
- Das Protokoll UPnP IGD
- Das graphische Webinterface

Wenn Anwendungen Dritter auf die Konfiguration oder auf Live-Werte der FRITZ!Box zugreifen wollen, wird stark empfohlen, dies über die standardisierten Verfahren TR-064 oder UPnP IGD zu realisieren. Auf www.avm.de/schnittstellen sind umfangreiche Dokumentationen zu diesen beiden Protokollen verfügbar.

Sollte dennoch gewünscht sein, auf das FRITZ!Box-Webinterface zuzugreifen, obwohl dort weder Abwärtskompatibilität noch Konsistenz zugesichert werden, wird im Folgenden beschrieben, wie ein Login zum Erhalten einer Session-ID funktioniert.

## Anmeldearten aus dem Heimnetz

Vor FRITZ!OS 7.24 konnte die Anmeldung am FRITZ!Box-Webinterface aus dem Heimnetz auf drei Arten erfolgen:

- Mit Benutzernamen und Kennwort
- Nur mit FRITZ!Box-Kennwort (Auslieferungszustand, FRITZ!Box-Kennwort auf Aufkleber auf der Unterschale des Gerätes)
- Ohne Kennwort (nicht empfohlen)

**Ab FRITZ!OS 7.24 wurden die Anmeldearten reduziert. Auf technischer Ebene erfolgt eine Anmeldung aus dem Heimnetz ab FRITZ!OS 7.24 nur noch mit Benutzernamen und Kennwort. Weiterführende Hinweise und Empfehlungen zu Benutzerdialogen für den Anmeldeprozess finden Sie unter https://avm.de/Schnittstellen im Dokument „Empfehlungen zur Benutzerführung bei der Anmeldung an einer FRITZ!Box“.**

Für die Anmeldung aus dem Internet wird ausschließlich die Anmeldung mit Benutzernamen und Kennwort unterstützt, auch in Versionen vor 7.24

## Einstiegsseite für Drittanwendungen

Informationen zum genutzten Login-Verfahren sowie die Vergabe der Session-ID erfolgt ab FRITZ!OS 5.50 über ein http-GET auf die Einstiegsseite

```
https://fritz.box/login_sid.lua?version=2
```

Hinweis: Sollte die Seite nicht aufgerufen werden können, handelt es sich um eine alte Firmware, die keine Session-IDs unterstützt.

Je nach übergebenen POST-Parametern können folgende Aktionen ausgeführt werden:

1) Eine Anmeldung durchführen.
   Parameter:
   "username" (Name des Benutzers oder leer),

"response" (eine aus Kennwort und Challenge erzeugte Response)

2) Prüfung, ob eine bestimmte Session-ID gültig ist.
   Parameter:
   "sid" (die zu prüfende Session-ID)

3) Abmelden, d.h. eine Session-ID ungültig machen.
   Parameter:
   "logout", "sid" (eine gültige Session-ID)

Die Antwort-XML-Datei hat immer den gleichen Aufbau:

```
<SessionInfo>
      <SID>8b4994376ab804ca</SID>
      <Challenge>a0fa42bb</Challenge>
      <BlockTime>0</BlockTime>
      <Users>
            <User last=1>fritz1337</User>
      </Users>
      <Rights>
            <Name>NAS</Name>
            <Access>2</Access>
            <Name>App</Name>
            <Access>2</Access>
            <Name>HomeAuto</Name>
            <Access>2</Access>
            <Name>BoxAdmin</Name>
            <Access>2</Access>
            <Name>Phone</Name>
            <Access>2</Access>
      </Rights>
</SessionInfo>
```

Die Anzahl und Reihenfolge der Werte im Bereich ist variabel. Es werden nur die Rechte geliefert, die die aktuelle Session-ID auch tatsächlich hat.

Bedeutung der Werte:

- <SID>: Besteht dieser Wert nur aus Nullen, bestehen keinerlei Rechte. Eine Anmeldung ist erforderlich.
  Ansonsten enthält man eine gültige Session-ID für den weiteren Zugriff auf die FRITZ!Box. Die aktuellen Zugriffsrechte stehen im Bereich <Rights>.
- <Challenge>: Enthält eine Challenge, mit der über das Challenge-Response-Verfahren eine Anmeldung durchgeführt werden kann.
- <BlockTime>: Zeit in Sekunden, in der kein weiterer Anmeldeversuch zugelassen wird.
- <Users>: Eine Liste aller User. Nur im Heimnetz abrufbar, falls aktiviert. Das Attribut „last=1“ Markiert den Nutzer, der zuletzt eingeloggt war, ansonsten ist „last=0“.
- <Rights>: Die einzelnen Rechte, die die aktuelle Session-ID hat. Mögliche Werte sind 1 (nur lesen) und 2 (Lese- und Schreibzugriff).

## Erhalt einer Session-ID

Ist eine Anmeldung erforderlich, muss zunächst im Rahmen des Logins eine gültige Session-ID erzeugt werden. Aus Sicherheitsgründen erfolgt das Login nicht mit dem Klartextkennwort, sondern über ein Challenge-Response-Verfahren.

### Ermittlung des Response-Wertes

#### *PBKDF2 (ab FRITZ!OS 7.24)*

Ab FRITZ!OS 7.24 kann ein Client durch Anhängen von `version=2` an den initialen GET-Request das moderne PBKDF2-basierte Challenge-Response-Verfahren anfordern. Beginnt die `<Challenge>` mit „`2$`“, so wird PBKDF2 von der FRITZ!OS Version unterstützt. Ansonsten ist ein Fallback auf MD5 möglich. Drittanwendungen wird empfohlen, PBKDF2 zu unterstützen, da die Kennwörter der Nutzer dadurch besser gegen Angriffe geschützt sind.

Die `<challenge>` wird aus der Einstiegsseite login_sid.lua?version=2 ausgelesen.

Das Format der Challenge ist folgendermaßen definiert, getrennt durch $-Zeichen:

```
2$<iter1>$<salt1>$<iter2>$<salt2>
```

Die Response wird folgendermaßen gebildet:

```
<hash1> = pbkdf2_hmac_sha256(<password>, <salt1>, <iter1>)
<response> = <salt2>$ + pbkdf2_hmac_sha256(<hash1>, <salt2>, <iter2>)
```

Für die Beispiel Challenge „`2$10000$5A1711$2000$5A1722`“ und das Passwort „`1example!`“ (utf8-encodiert) ergibt sich also:

```
hash1 = pbkdf2_hmac_sha256("1example!", 5A1711, 10000)
=> 0x23428e9dec39d95ac7a40514062df0f9e94f996e17c398c79898d0403b332d3b (hex)

response = 5A1722$ + pbkdf2_hmac_sha256(hash1, 5A1722, 2000).hex()
=> 5A1722$1798a1672bca7c6463d6b245f82b53703b0f50813401b03e4045a5861e689adb
```

Die Zahlen `iter1` und `iter2` sind der jeweilige Iterationszähler Parameter für pbkdf2 der 1. und 2. Runde. Diese sind variabel gehalten, um den Login über PBKDF2 zukunftssicher zu gestalten.

Durch das doppelte Hashen, mit jeweils anderem Salt, kann ein Teil der Antwort in FRITZ!OS statisch vorberechnet werden, wodurch ein Angriff schwerer wird, der Login für Benutzer aber gewohnt schnell bleiben kann.

Der Login-Vorgang geschieht nun über einen POST-Request an login_sid.lua?version=2 mit „`response`“ als Parameter.

Die Antwort-XML der FRITZ!Box beinhaltet nun die gültige SID.

Beispielcode findet sich unten, in der Funktion `calculate_pbkdf2_response`.

#### MD5

Beginnt die Challenge nicht mit `2$`, da es sich um eine ältere FRITZ!OS Version handelt, oder wird der Request an login_sid.lua ohne den `version=2` Parameter gestellt, so wird (seit FRITZ!OS 5.50) das MD5 Digest Verfahren verwendet.

Der Response-Wert wird aus dem Klartextkennwort und einer Challenge wie folgt ermittelt:

`<challenge>` wird aus der Einstiegsseite login_sid.lua?version=2 ausgelesen.

`<md5>` ist der MD5 (`<challenge>-<klartextpasswort>`) in 32 Hexzeichen mit Kleinbuchstaben

Der MD5-Hash wird über die Bytefolge der UTF-16LE-Codierung dieses Strings gebildet (ohne BOM und ohne abschließende 0-Bytes).

Aus Kompatibilitätsgründen muss für jedes Zeichen, dessen Unicode Codepoint > 255 ist, die Codierung des "."-Zeichens benutzt werden (0x2e 0x00 in UTF-16LE). Dies betrifft also alle Zeichen, die nicht in ISO-8859-1 dargestellt werden können, z. B. das Euro-Zeichen.

Beispiel mit deutschem Umlaut:

Die Challenge

```
<challenge> = "1234567z"
```

kombiniert mit dem Kennwort

```
<password> = "äbc"
```

ergibt den Wert

```
<response> = "1234567z-9e224a41eeefa284df7bb0f26c2913e2"
```

Der Login-Vorgang geschieht nun über einen POST-Request an login_sid.lua?version=2 mit „response“ als Parameter.

Die Antwort-XML der FRITZ!Box beinhaltet nun die gültige SID.

Beispielcode findet sich unten, in der Funktion `calculate_md5_response`.

## Verwendung der Session-ID

Die Session-ID ist eine 64-Bit-Zahl, die durch 16 Hexziffern dargestellt wird. Sie wird beim Login vergeben und muss für die Dauer der Sitzung mitgeführt werden. Dabei sollte ein Programm zu jeder FRITZ!Box jeweils nur eine Session-ID verwenden, da die Anzahl der Sessions zu einer FRITZ!Box beschränkt ist.

Die Session-ID hat nach Vergabe eine Gültigkeit von 20 Minuten. Die Gültigkeitsdauer verlängert sich automatisch bei aktivem Zugriff auf die FRITZ!Box.

Die Session-ID 0 (`0000000000000000`) ist immer ungültig.

Die Übergabe der Session-ID erfolgt im Parameter "sid".

### Zugriff ohne Session-ID

Grundsätzlich können alle dynamisch generierten Seiten nur mit einer gültigen Session-ID aufgerufen werden. Auch das Lesen oder Schreiben von Web-Variablen erfordert eine Session-ID.

Folgende Inhalte können ohne gültige Session-ID aufgerufen werden:

- Einstiegsseiten (z. B. Login-Seite)
- Statische Inhalte (z. B. Grafiken)

### Beenden einer Sitzung

Eine Sitzung kann durch Löschen der Session-ID jederzeit auch vor Ablauf des automatischen Timeouts beendet werden.

Dies geschieht durch Aufruf der login_sid.lua?version=2 mit den Parametern „`logout`" und der aktuellen SID.

## Beispiel-Code für den Bezug einer Session-ID

### Beispiel-Code Python3

```
#!/usr/bin/env python3
# vim: expandtab sw=4 ts=4

"""
FRITZ!OS WebGUI Login

Get a sid (session ID) via PBKDF2 based challenge response algorithm.
Fallback to MD5 if FRITZ!OS has no PBKDF2 support.
AVM 2020-09-25
"""

import sys
import hashlib
import time
import urllib.request
import urllib.parse
import xml.etree.ElementTree as ET

LOGIN_SID_ROUTE = "/login_sid.lua?version=2"


class LoginState:
    def __init__(self, challenge: str, blocktime: int):
        self.challenge = challenge
        self.blocktime = blocktime
        self.is_pbkdf2 = challenge.startswith("2$")


def get_sid(box_url: str, username: str, password: str) -> str:
    """ Get a sid by solving the PBKDF2 (or MD5) challenge-response
process. """
    try:
        state = get_login_state(box_url)
    except Exception as ex:
        raise Exception("failed to get challenge") from ex

    if state.is_pbkdf2:
        print("PBKDF2 supported")
        challenge_response = calculate_pbkdf2_response(state.challenge,
password)
    else:
        print("Falling back to MD5")
        challenge_response = calculate_md5_response(state.challenge,
password)

    if state.blocktime > 0:
        print(f"Waiting for {state.blocktime} seconds...")
        time.sleep(state.blocktime)

    try:
        sid = send_response(box_url, username, challenge_response)
    except Exception as ex:
        raise Exception("failed to login") from ex
    if sid == "0000000000000000":
        raise Exception("wrong username or password")
    return sid


def get_login_state(box_url: str) -> LoginState:
```

```
    """ Get login state from FRITZ!Box using login_sid.lua?version=2 """
    url = box_url + LOGIN_SID_ROUTE
    http_response = urllib.request.urlopen(url)
    xml = ET.fromstring(http_response.read())
    # print(f"xml: {xml}")
    challenge = xml.find("Challenge").text
    blocktime = int(xml.find("BlockTime").text)
    return LoginState(challenge, blocktime)


def calculate_pbkdf2_response(challenge: str, password: str) -> str:
    """ Calculate the response for a given challenge via PBKDF2 """
    challenge_parts = challenge.split("$")
    # Extract all necessary values encoded into the challenge
    iter1 = int(challenge_parts[1])
    salt1 = bytes.fromhex(challenge_parts[2])
    iter2 = int(challenge_parts[3])
    salt2 = bytes.fromhex(challenge_parts[4])
    # Hash twice, once with static salt...
    hash1 = hashlib.pbkdf2_hmac("sha256", password.encode(), salt1, iter1)
    # Once with dynamic salt.
    hash2 = hashlib.pbkdf2_hmac("sha256", hash1, salt2, iter2)
    return f"{challenge_parts[4]}${hash2.hex()}"


def calculate_md5_response(challenge: str, password: str) -> str:
    """ Calculate the response for a challenge using legacy MD5 """
    response = challenge + "-" + password
    # the legacy response needs utf_16_le encoding
    response = response.encode("utf_16_le")
    md5_sum = hashlib.md5()
    md5_sum.update(response)
    response = challenge + "-" + md5_sum.hexdigest()
    return response


def send_response(box_url: str, username: str, challenge_response: str) ->
str:
    """ Send the response and return the parsed sid. raises an Exception on
error """
    # Build response params
    post_data_dict = {"username": username, "response": challenge_response}
    post_data = urllib.parse.urlencode(post_data_dict).encode()
    headers = {"Content-Type": "application/x-www-form-urlencoded"}
    url = box_url + LOGIN_SID_ROUTE
    # Send response
    http_request = urllib.request.Request(url, post_data, headers)
    http_response = urllib.request.urlopen(http_request)
    # Parse SID from resulting XML.
    xml = ET.fromstring(http_response.read())
    return xml.find("SID").text


def main():
    if len(sys.argv) < 4:
        print(
            f"Usage: {sys.argv[0]} http://fritz.box user pass"
        )
        exit(1)

    url = sys.argv[1]
    username = sys.argv[2]
    password = sys.argv[3]
```

```
    sid = get_sid(url, username, password)
    print(f"Successful login for user: {username}")
    print(f"sid: {sid}")


if __name__ == "__main__":
    main()
```

## Beispielcode Java

```
import java.nio.charset.StandardCharsets;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class Pbkdf2Login {

    /**
     * Calculate the secret key on Android.
     */
    public static String calculatePbkdf2Response(String challenge, String
password) {
        String[] challenge_parts = challenge.split("\\$");
        int iter1 = Integer.parseInt(challenge_parts[1]);
        byte[] salt1 = fromHex(challenge_parts[2]);
        int iter2 = Integer.parseInt(challenge_parts[3]);
        byte[] salt2 = fromHex(challenge_parts[4]);
        byte[] hash1 = pbkdf2HmacSha256(password.getBytes(Standard-
Charsets.UTF_8), salt1, iter1);
        byte[] hash2 = pbkdf2HmacSha256(hash1, salt2, iter2);
        return challenge_parts[4] + "$" + toHex(hash2);
    }

    /** Hex string to bytes */
    static byte[] fromHex(String hexString) {
        int len = hexString.length() / 2;
        byte[] ret = new byte[len];
        for (int i = 0; i < len; i++) {
            ret[i] = (byte) Short.parseShort(hexString.substring(i * 2, i *
2 + 2), 16);
        }
        return ret;
    }

    /** byte array to hex string */
    static String toHex(byte[] bytes) {
        StringBuilder s = new StringBuilder(bytes.length * 2);
        for (byte b : bytes) { s.append(String.format("%02x", b)); }
        return s.toString();
    }

    /**
     * Create a pbkdf2 HMAC by appling the Hmac iter times as specified.
     * We can't use the Android-internal PBKDF2 here, as it only accepts
char[] arrays, not bytes (for multi-stage hashing)
     */
    static byte[] pbkdf2HmacSha256(final byte[] password, final byte[]
salt, int iters) {
        try {
            String alg = "HmacSHA256";
```

```
            Mac sha256mac = Mac.getInstance(alg);
            sha256mac.init(new SecretKeySpec(password, alg));
            byte[] ret = new byte[sha256mac.getMacLength()];
            byte[] tmp = new byte[salt.length + 4];
            System.arraycopy(salt, 0, tmp, 0, salt.length);
            tmp[salt.length + 3] = 1;
            for (int i = 0; i < iters; i++) {
                tmp = sha256mac.doFinal(tmp);
                for (int k = 0; k < ret.length; k++) { ret[k] ^= tmp[k]; }
            }
            return ret;
        } catch (NoSuchAlgorithmException | InvalidKeyException e) {
            return null; // TODO: Handle this properly
        }
    }
}
```