APPENDIX

A PROPERTIES EXTRACTED FROM EXISTING CVES

Table 5: Properties extracted from existing CVEs in the implementations of Pro-FTPD (*PrF*) for the FTP protocol, Live555 (*LV*) for the RTSP protocol, OpenSSH (*SH*) for the SSH protocol, OpenSSL (*SL*) for the TLS protocol, TinyDTLS (*TD*) for the DTLS protocol, Contiki-Telnet (*CT*) for the TELNET protocol, and Pure-FTPd (*PuF*) for the FTP protocol.

ID	Vulnerability	Property Description	LTL Notation
PrF_1	CVE-2019-18217	After one client succeeds to connect with a sever, the server should finally give responses for requests from the connected client.	G((LogIN) → (X(G((Requests) → (X(F(Responses)))))))
PrF_2	CVE-2019-12815	If a client does not log in successfully, the server must not allow this client to copy files.	G((¬(LogIN)) → (X(G((request = CopyFiles) → X(¬(response = CopySuccessful))))))
PrF_3	CVE-2015-3306	If the server receives CPTO requests when the client doesn't succeed to log in, must not allow CPTO successfully.	G((¬(LogIN)) → (X(G((request = CPTO) → X(¬(response = CPTOSuccessful))))))
PrF_4	CVE-2010-3867	If the client logs in and is only assigned one writable directory, the server must not allow it to write out of scope of the assigned directory.	G(((state = LogIN) ∧ (WritableDirectory = true) ∧ (request = OverWrite)) → (X(response = PermissionDenied)))
LV_1	CVE-2019-6256	After the connection channel is DESTROYED between the server and the client, the channel must not be USED unless one new connection is ESTABLISHED.	G((channel = DESTROYED) → (X(((channel = ESTABLISHED) R (¬(channel = USED))))))
LV_2	CVE-2019-15232	The server must not create two client sessions with the same ID.	G((SessionID = RID) → (X(G(¬(SessionID = RID)))))
LV_3	CVE-2019-7314	If the server receives the PLAY request in the INIT state, must not begin StartPlay	G(((state = INIT) ∧ (request = PLAY)) → (X(¬(response = StartPlay))))
LV_4	CVE-2013-6934	If receiving a invalid request, must always refuse it with Method_not_Allowed.	G((request = InvalidRequest) → X(response = Method_not_Allowed))
LV_5	CVE-2013-6933	If receiving an invalid request, must always refuse it with Method_not_Allowed.	G((request = InvalidRequest) → X(response = Method_not_Allowed))
SH_1	CVE-2018-15473	Whenever the server receives invalid username or valid username with wrong password, must give the same response.	G(((request = InvalidUsername) ∨ (request = ValidUsername&WrongPasswd)) → (X(G(SameResponse))))
SH_2	CVE-2016-6210	Whenever the server receives invalid username or valid username with wrong password, must give responses within the same time period.	G(((request = InvalidUsername) ∨ (request = ValidUsername&WrongPasswd)) → (X(G(SameTimeToResponse))))
SL_1	CVE-2016-6309	If the server receives the ChangeCipherSpec request after sending the ServerHello response, should give a ChangeCipherSpec response or an Alert.	G((response = ServerHello) → X((request = ChangeCipherSpec) → X((response = ChangeCipherSpec) ∨ (response = Alert))))
SL_2	CVE-2016-6305	If the server receives an ApplicationData after the Handshake is successful, must finally give an Alert or response the ApplicationData.	G((state = HandshakeDone) ∧ (request = ApplicationData) → (X(F(response = ApplicationData) ∨ (response = Alert)))))
SL_3	CVE-2014-0160	After the server receives a ClientHello request with the Heartbeat_extension in the peer_allowed_to_send mode, and gives a ServerHello response with the same options, the sever receives a malformed Heartbeat request with the payload length field number larger than the real payload length, must always not send Heartbeat responses.	G((((request = ClientHello) ∧ (Heartbeat_extension = true) ∧ (peer_allowed_to_send = 1)) ∧ X((response = ServerHello) ∧ (Heartbeat_extension = true) ∧ (peer_allowed_to_send = 1))) → F(G(((((request = Heartbeat_Request) ∧ (Payload_Length > realPayloadLength))) → F(G(¬(response = Heartbeat_Response))))))

B LTL PROPERTIES EXTRACTED FROM RFC AND COMMENTS

Please see the following pages for the Linear-time Temporal Logic properties extracted from sources such as RFCs.

Table 6: Properties extracted from relevant RFCs of network protocols and comments in the implementations of Pro-FTPD (PrF) for the FTP protocol, Live555 (LV) for the RTSP protocol, OpenSSH (SH) for the SSH protocol, OpenSSL (SL) for the TLS protocol, TinyDTLS (TD) for the DTLS protocol, Contiki-Telnet (CT) for the TELNET protocol, and Pure-FTPd (PuF) for the FTP protocol.

NO	PID	Property Description	LTL Notation
1	PrF_5	If receiving invalid username or invalid password, the server must always show the same message to the user.	G(((request = InvalidUsername) ∨ (request = InvalidPassword)) → X(G(sameResponse)))
2	PrF_6	If receiving the CWD request without login, the server must not give the CommandOkay response.	G((¬(state = LogIN) ∧ (request = CWD)) → X(G(¬(response = CommandOkay))))
3	PrF_7	After a connection is constructed successfully, there should be a successful login and after that without failed login.	G(((request = ValidUserName&ValidPasswd) → X(response = LoginSuccess)) → X(G(¬(response = LoginFailed))))
4	PrF_8	After the connection is lost after a long time, responses should be always timeout.	G(LostConnection → X(G(response = Timeout)))
5	LV_6	If the server is in the Play state and receives a Pause request, should go into the Ready state.	G(((state = Play) ∧ (request = Pause)) → X(state = Ready))
6	LV_7	If the server is in the Play state and receives a TEARDOWN request, should go into the Init state.	G(((state = Play) ∧ (request = TEARDOWN)) → X(state = Init))
7	LV_8	If the server is in the Ready state and receives a Play request with one old URI, should response ChangeTransportParam.	G(((state = Ready) ∧ (request = Play) ∧ (OldURI = true)) → X(response = ChangeTransportParam))
8	LV_9	If the server is connected with a client and then receives a TEARDOWN request, should finally give a TeardownSuccess or Timeout response.	G((((request = Setup) ∧ X(response = SetupSuccess)) ∧ X(request = TEARDOWN)) → X(F((response = TeardownSuccess) ∨ (response = Timeout))))
9	LV_{10}	The TEARDOWN request will not be acknowledged until the SETUP request is be acknowledged.	G(¬((request = TEARDOWN) ∧ X(response = TeardownSuccess)) U ((request = SETUP) ∧ X(response = SetupSuccess)))
10	SH_3	If the server receives the SSH_MSG_CHANNEL_OPEN request and gives a SSH_MSG_CHANNEL_OPEN_ CONFIRMATION response, and then receives a Login request and gives a SSH_MSG_USERAUTH_SUCCESS, there will not have a failure in user authentication.	G(((request = SSH_MSG_CHANNEL_OPEN) ∧ X(response = SSH_MSG_CHANNEL_OPEN_CONFIRMATION) ∧ X(request = Login) ∧ X(response = SSH_MSG_USERAUTH_SUCCESS)) → X(G(¬(response = SSH_MSG))))
11	SH_4	After the server gives a KEXINIT response, will not give the KEXINIT or AcceptConnection response until receiving the NewKeys request.	G((response = KEXINIT) → X((¬(response = KEXINIT) ∧ ¬(response = AcceptConnection)) U (request = NewKeys)))
12	SH_5	All authentication messages after a ConnectionSuccess response should give no response until the end condition is true.	G((request = ConnectionSuccess) → X((NoResponse) U (EndCondition = true)))
13	SL_4	If SECURE_RENEGOTIATION is disabled and the server receives a ClientHello request with renegotiation option and an empty "RENEGOTIATED_CONNECTION" field, must send a ServerHello response without the renegotiation option.	G((SECURE_RENEGOTIATION = disabled) ∧ (request = ClientHello) ∧ (RenegotiationExtension = enabled) ∧ (RENEGOTIATED_CONNECTION = empty) → X((response = ServerHello) ∧ (RenegotiationExtension = disabled)))
14	SL_5	If SECURE_RENEGOTIATION is disabled the server receives a ClientHello with renegotiation extension and not an empty "RENEGOTIATED_CONNECTION" field, must give a HandshakeFailure response.	G((SECURE_RENEGOTIATION = disabled) ∧ (request = ClientHello) ∧ (RenegotiationExtension = enabled) ∧ (¬(RENEGOTIATED_CONNECTION = empty))) → X(response = HandshakeFailure)))
15	SL_6	If SECURE_RENEGOTIATION is enabled and the server receive ClientHello with SCSV (TLS_EMPTY_RENEGOTIATION_INFO_SCSV), must give a HandshakeFailure response.	G((SECURE_RENEGOTIATION = enabled) ∧ (request = ClientHello) ∧ (SCSV= enabled) → X(response = HandshakeFailure))
16	SL_7	If SECURE_RENEGOTIATION is enabled and the server receives a ClientHello request without renegotiation extension, must then abort the handshake with a HandshakeFailure response.	G((SECURE_RENEGOTIATION = enabled) ∧ (request = ClientHello) ∧ (RenegotiationExtension = false) → X(response = HandshakeFailure))
17	SL_8	If SECURE_RENEGOTIATION is enabled and the server receives ClientHello request but "RENEGOTIATED_CONNECTION" field is not the same as the saved CLIENT_VERIFY_DATA value, must give a HandshakeFailure response.	G((SECURE_RENEGOTIATION = enabled) ∧ (request = ClientHello) ∧ (¬(RENEGOTIATED_CONNECTION = CLIENT_VERIFY_DATA))) → X(response = HandshakeFailure))
18	SL_9	After the server receives a ClientHello request without renegotiation extension and gives a ServerHello response, then receives a ClientHello again, must refuse the renegotiation with an Alert.	G(((((request = ClientHello) ∧ (RenegotiationExtension = disabled)) ∧ (X(response = ServerHello))) ∧ (X(request = ClientHello))) → X(response = Alert))

19	TD_1	If the server is in the WAIT_CLIENTHELLO state and receives a ClientHello request with valid cookie and the epoch value 0, must finally give ServerHello responses.	G(((state = WAIT_CLINETHELLO) ∧ (request = ClientHello) ∧ (ValidCookie = true) ∧ (EpochValue = 0)) ⟶ X(F(response = ServerHello)))
20	TD_2	If the server is in the WAIT_CLIENTHELLO state and receives a ClientHello request with valid cookie but not 0 epoch value, must not give ServerHello responses.	G(((state = WAIT_CLINETHELLO) ∧ (request = ClientHello) ∧ (ValidCookie = true) ∧ (¬(EpochValue = 0))) ⟶ X(¬(response = ServerHello)))
21	TD_3	If the server is in the WAIT_CLIENTHELLO state and receives a ClientHello request with an invalid cookie, must reply HelloVerifyRequest.	G(((state = WAIT_CLIENTHELLO) ∧ (request = ClientHello) ∧ (ValidCookie = false)) ⟶ X(response = HelloVerifyRequest))
22	TD_4	If the server is in the WAIT_CLIENTHELLO state but receives a ChangeCipher request, must refuse it with an Alert.	G(((state = WAIT_CLIENTHELLO) ∧ (request = ChangeCipher)) ⟶ X(response = Alert))
23	TD_5	If the server is in the DTLS_HT_CERTIFICATE_REQUEST state and receives a Certificate request, must give a DTLS_ALERT_HANDSHAKE_FAILURE response or DTLS_ALERT_DECODE_ERROR, or set Client_Auth to be verified.	G(((state = DTLS_HT_CERTIFICATE_REQUEST) ∧ (request = Certificate)) ⟶ X((response = DTLS_ALERT_DECODE_ERROR) ∨ (response = DTLS_ALERT_HANDSHAKE_FAILURE) ∨ (Client_Auth = true)))
24	TD_6	If SECURE_RENEGOTIATION is disabled and the server receives a ClientHello request with renegotiation option and an empty "RENEGOTIATED_CONNECTION" field, must send a ServerHello response without the renegotiation option.	G((SECURE_RENEGOTIATION = disabled) ∧ (request = ClientHello) ∧ (RenegotiationExtension = enabled) ∧ (RENEGOTIATED_CONNECTION = empty) ⟶ X((response = ServerHello) ∧ (RenegotiationExtension = disabled))))
25	TD_7	If SECURE_RENEGOTIATION is disabled the server receives a ClientHello with renegotiation extension and not an empty "RENEGOTIATED_CONNECTION" field, must give a HandshakeFailure response.	G(((SECURE_RENEGOTIATION = disabled) ∧ (request = ClientHello) ∧ (RenegotiationExtension = enabled) ∧ (¬(RENEGOTIATED_CONNECTION = empty))) ⟶ X(response = HandshakeFailure))
26	TD_8	If SECURE_RENEGOTIATION is enabled and the server receive ClientHello with SCSV (TLS_EMPTY_RENEGOTIATION_INFO_SCSV), must give a HandshakeFailure response.	G((SECURE_RENEGOTIATION = enabled) ∧ (request = ClientHello) ∧ (SCSV= enabled) ⟶ X(response = HandshakeFailure))
27	TD_9	If SECURE_RENEGOTIATION is enabled and the server receives a ClientHello request without renegotiation extension, must then abort the handshake with a HandshakeFailure response.	G((SECURE_RENEGOTIATION = enabled) ∧ (request = ClientHello) ∧ (RenegotiationExtension = false) ⟶ X(response = HandshakeFailure))
28	TD_{10}	If SECURE_RENEGOTIATION is enabled and the server receives ClientHello request but "RENEGOTIATED_CONNECTION" field is not the same as the saved CLIENT_VERIFY_DATA value, must give a HandshakeFailure response.	G(((SECURE_RENEGOTIATION = enabled) ∧ (request = ClientHello) ∧ (¬(RENEGOTIATED_CONNECTION = CLIENT_VERIFY_DATA))) ⟶ X(response = HandshakeFailure))
29	TD_{11}	After the server receives a ClientHello request without renegotiation extension and gives a ServerHello response, then receives a ClientHello again, must refuse the renegotiation with an Alert.	G(((((request = ClientHello) ∧ (RenegotiationExtension = disabled)) ∧ (X(response = ServerHello))) ∧ (X(request = ClientHello))) ⟶ X(response = Alert))
30	TD_{12}	After the server receives a ClientHello request and gives a ServerHello response, then receives a ClientKeyExchange request with a different epoch value than that of ClientHello, server must not give ChangeCipherSpec responses.	G(((((request = ClientHello) ∧ X(response = ServerHello)) ∧ (X((request = ClientKeyExchange) ∧ (¬(EpochValue$_{cke}$ = EpochValue$_{ch}$))))) ⟶ X(¬(response = ChangeCipherSpec)))
31	TD_{13}	After the server receives a ClientHello request and gives a ServerHello response, then receives a ClientHello request with the same epoch value as that of the first one, server must not give ServerHello.	G((request = ClientHello) ∧ (X(response = ServerHello)) ∧ (X((request = ClientHello) ∧ (EpochValue$_{c1}$ = EpochValue$_{c2}$))) ⟶ (X(¬(response = ServerHello))))
32	TD_{14}	If the server receives a ClientHello request and gives a HelloVerifyRequest response, and then receives a over-large packet even with valid cookies, the server must refuse it with an Alert.	G(((request = ClientHello) ∧ X(response = HelloVerifyRequest)) ⟶ X(G((request = OverLargePacket) ⟶ X(response = Alert))))
33	CT_1	After WILL request is received and the corresponding option is disabled, must send DO or DONT responses.	G(((request = WILL) ∧ (option = Disabled)) ⟶ X((response = DO) ∨ (response = DONT)))
34	CT_2	After DO request is received and the corresponding option is disabled, must send WILL or WONT responses.	G(((request = DO) ∧ (option = Disabled)) ⟶ X((response = WILL) ∨ (response = WONT)))
35	CT_3	After WILL request is received and the corresponding option is enabled, must not give responses.	G(((request = WILL) ∧ (option = Enabled)) ⟶ X(¬(Response)))
36	CT_4	After DO request is received and the corresponding option is enabled, must not give responses.	G(((request = DO) ∧ (option = Enabled)) ⟶ X(¬(Response)))
37	CT_5	After WONT request is received and the corresponding option is enabled, must send the DONT response.	G(((request = WONT) ∧ (option = Enabled)) ⟶ X(response = DONT))

38	CT_6	After DONT request is received and the corresponding option is enabled, must send the WONT response.	G(((request = DONT) ∧ (option = Enabled)) → X(response = WONT))
39	CT_7	After WONT request is received and the corresponding option is disabled, must not give responses.	G(((request = WONT) ∧ (option = Disabled)) → X(¬(Response)))
40	CT_8	After DONT request is received and the corresponding option is disabled, must not give responses.	G(((request = DONT) ∧ (option = Disabled)) → X(¬(Response)))
41	CT_9	If receive IAC in NORMAL state, next go to the SIAC state and finally go back to the NORMAL state	G((((request = IAC) ∧ (state = NORMAL)) → X(G((state = IAC))) → X(F(state = NORMAL)))))
42	CT_{10}	Before Disconnection, must send an Alert to disconnect with clients.	G((¬(Disconnection)) ∪ (response = Alert))
43	CT_{11}	If conduct COMMAND without AbortOutput, the response must be same as the real execution results.	G(((request = COMMAND) ∧ (¬(AbortOutput))) → X(G(response = realResults)))
44	PuF_1	If receiving invalid username or invalid password, the server must always show the same message to the user.	G(((request = InvalidUsername) ∨ (request = InvalidPassword)) → X(G(sameResponse)))
45	PuF_2	After one client succeeds to connect with a sever, the server should finally give responses for requests from the connected client.	G((LogIN) → (X(G((Requests) → (X(F(Responses)))))))
46	PuF_3	If receiving the CWD request without login, the server must not give the CommandOkay response.	G((¬(state = LogIN) ∧ (request = CWD)) → X(G(¬(response = CommandOkay))))
47	PuF_4	If a client doesn't log in successfully, the server must not allow this client to copy files.	G((¬(LogIN)) → (X(G((request = CopyFiles) → X(¬(response = CopySuccessful))))))
48	PuF_5	If user directory size is larger than the set quota when the quota mechanism is activated, must finally reply that the quota is exceeded.	¬F((quota_activated = true) ∧ F((user_dir_size > user_quota)∧ G(¬(msg_quota_exceeded= true))))
49	PuF_6	After a connection is constructed successfully, there should be a successful login and after that without failed login.	G((((request = ValidUserName) ∧ X(request = ValidPasswd)) ∧ X(response = LoginSuccess)) → X(G(¬(response = LoginFailed))))
50	PuF_7	After the connection is lost after a long time, responses should be always timeout.	G(LostConnection → X(G(response = Timeout)))