Valkyria DMTP-2.01

Protokół komunikacyjny serwera

1. Budowa protokołu

Protokół DMTP w wersji 2.01 ma zastosowanie do przesyłu wiadomości zaszyfrowanych od końca do końca (ang. end to end, E2E). Protokół ten wprowadza dodatkowe szyfrowanie i uwierzytelnienie pomiędzy serwerem i klientem w taki sposób, że po pierwszym nawiązaniu połączenia z serwerem nie jest możliwy udany atak MITM, o ile tożsamość kryptograficzna serwera została zapamiętana w aplikacji łączącej się.

Protokół DMTP-2.01 może być stosowany bezpośrednio na warstwie fizycznej lub też na protokole UDP w warstwie transportowej (według modelu OSI). Przy stosowaniu UDP obowiązkowe jest stosowanie protokołu IP w warstwie sieci.

Protokół DMTP składa się z obsługi protokołu STUN/ICE zgodnego ze specyfikacjami RFC-5389 i RFC-8445 oraz DTLS opisanego w RFC 6347. Na warstwie DTLS zawsze znajduje się warstwa SCTP. SCTP posiada podstawową implementację wystarczającą do implementacji WebRTC w środowisku, w którym o spójność danych dba protokół DTLS. W warstwie SCTP zaimplementowany jest protokół DCEP (RFC 8832). protokół STUN/ICE jest dostępny także po nawiązaniu połączenia DTLS, co jest istotne dla nawiązywania połączeń bezpośrednich przez klientów między sobą, bowiem DMTP-2.01 umożliwia także nawiązywanie połączeń bezpośrednich klient do klienta (ang. peer to peer, P2P) w sytuacji, gdy przynajmniej jeden z klientów ma asymetryczny NAT (typu CONE).

Powyższa budowa protokołu wymuszona jest przez fakt, że przeglądarki internetowe jako protokół oparty na transmisji wyłącznie UDP oferują w 2025 roku jedynie rozwiązanie WebRTC ze stosem protokołów UDP/DTLS/SCTP/DCEP. Jako opcja dla aplikacji desktopowych przewidywane jest zaimplementowanie HDLC (typ 3) celem stworzenia mniej obciążającej warstwy transportowej.

W tym miejscu należy zaznaczyć, że DTLS wraz z SCTP i DCEP są używane jako warstwa wyłącznie transportowa zapewniająca integralność oraz dotarcie pakietów danych w kolejności. Właściwości szyfrujące DTLS nie są wykorzystywane. Na warstwie SCTP znajduje się warstwa aplikacji protokołu DMTP, która zapewnia uwierzytelnianie i szyfrowanie.

W protokole DCEP klient łączący się musi przekazać wartość tekstową wskazującą protokół komunikacyjny i jego wersję. Dla niniejszej wersji jest to ciąg 9 znaków «DMTP-2.01». Połączenia z wartością nierozpoznawaną zostaną odrzucone na etapie wskazywania protokołu lub też później podczas korzystania z rozszerzonych funkcji protokołu.

1.1. Uwagi dotyczące bezpieczeństwa

Aplikacje rozpowszechniane do komunikacji przez protokół DMTP powinny mieć zaimplementowaną bazę tożsamości kryptograficznych znanych serwerów, tak aby w przypadku próby podszycia było to niemożliwe już przy pierwszym użyciu. Nie istnieje żaden sposób dający 100% pewność braku podszycia się pod nieznany nowy serwer w sytuacji, gdy nie jest znana jego tożsamość kryptograficzna (niezmienny klucz publiczny). Stosowanie jednak kluczy publicznych do identyfikowania rozmówców i ich przekazywanie na etapie spotkania daje 100% pewność, że rozmowa odbywa się między tymi osobami, gdy wzajemnie przekazały sobie klucze publiczne. Powodzenie ataku MITM w takiej sytuacji jest niemożliwe, bowiem obie osoby sprawdzają nawzajem poprawność podpisu cyfrowego. Poprawnie podpisać wiadomość może tylko osoba posiadająca klucz prywatny.

Klucze prywatne serwerów powinny być przechowywane na oddzielnym urządzeniu (zalecane podłączenie przez UART), aby uniemożliwić włamanie celem ich zdobycia. Klucz 6144-bitowy w takim scenariuszu może być używany wiele lat bez ryzyka jego kompromitacji. Urządzenie podłączone przez UART nie powinno ujawniać klucza, a zamiast tego realizować operację odszyfruj oraz podpisz cyfrowo. Szyfrowanie i weryfikację podpisu cyfrowego można realizować bezpośrednio na serwerze z wykorzystaniem kluczy publicznych.

1.2. Szyfrowanie E2E

Protokół DMTP-2.01 nie zapewnia sam w sobie szyfrowania E2E. Służy on do transportu wcześniej zaszyfrowanych wiadomości metodą E2E. Klient DMTP najpierw szyfruje wiadomość, następnie przesyła wiadomość drugi raz zaszyfrowaną według specyfikacji niniejszego protokółu. Serwer odbiera dane i odczytuje wiadomość. Wiadomość ta jest wciąż zaszyfrowana kluczem odbiorcy i nie jest możliwe rozczytanie co znajduje się w wiadomości. Serwer widzi jedynie numer polecenia (co ma zrobić) oraz numer odbiorcy. Tak zaszyfrowaną wiadomość składa do skrytki odbiorczej adresata nie interpretując jej. Adresat pobiera wiadomość z serwera przez dodatkowo szyfrowany tunel komunikacyjny niniejszym protokołem i odszyfrowuje wiadomość swoim kluczem prywatnym.

Powyższe oznacza, że każda treści wysłane przez użytkownika, niebędące częścią specyfikacji tego protokołu są zaszyfrowane podwójnie.

Serwer na żadnym etapie nie tylko nie dowiaduje się o czym rozmawia nadawca z odbiorcą, ale dodatkowo nie ma żadnego mechanizmu, by móc się tego dowiedzieć.

Celem niniejszego protokołu jest ukrycie z kim rozmówca rozmawia używając komunikatora szyfrowanego od końca do końca oraz przechowywanie wiadomości podczas nieobecności rozmówcy. Identyfikatory osób oraz ich klucze publiczne są ukrywane przez protokół DMTP-2.01.

1.3. Podprotokół STUN/ICE

Zaimplementowany protokół jest zgodny z normami RFC-5389 i RFC-8445. Dokumenty te jednak definiują, że powinno być stosowane hasło oraz nazwa użytkownika. Zaimplementowana jest też wstecz zgodność z RFC-3489, dzięki czemu klient nieznający hasła może się dowiedzieć jaki ma adres IP i z jakiego portu nadaje zapytanie.

Nazwy użytkowników (ang. username), protokołu STUN muszą być liczbą zapisaną szesnastkowo (cyframi i małymi literami) o długości 32 znaków. Hasło jest również 32-znakowe i wyliczane jest według wzoru:

Password = $username^{257}$ mod 340282366920938463463374607431768211297.

Hasło jest również zapisane szesnastkowo.

Serwer jest zawsze w trybie CONTROLLED i używa implementacji «lite». Klient musi więc implementować wersję protokołu STUN «full» oraz obsługiwać tryb CONTROLLING. W takim trybie działają zazwyczaj przeglądarki internetowe.

Hasło i nazwy użytkowników STUN pozostają więc jawne, ale protokół STUN nie służy do przesyłu innych informacji niż adres IP i port, które muszą pozostać jawne.

2. Warstwa aplikacji protokołu DMTP

2.1. Algorytmy stosowane w warstwie aplikacji

W warstwie aplikacji protokołu DMTP są stosowane algorytmy kryptograficzne podane niżej.

AES-256-CBC – standardowy algorytm szyfrujący opisany w dokumencie RFC-3602. Jedyną różnicą jest brak jawnego podawania wektora inicjującego IV w komunikatach. Dane, IV oraz klucze są zapisywane w notacji big endian.

Rabin-6144 – kryptosystem Rabina bazuje na trudności znalezienie pierwiastka kwadratowego modulo duża liczba złożona. Niezbędna do tego jest faktoryzacja klucza publicznego. Klucze, wiadomość jawna i szyfrogramy zapisane są w notacji big endian na 768 bajtach. Na etapie szyfrowania sprawdzane jest czy wartość poddawana szyfrowaniu jest parzysta i jednocześnie dodatnia i mniejsza od N/2 (gdzie N stanowi klucz publiczny). Sprawdzany jest także symbol Jacobi (względem N) i jeżeli wynik wynosi –1 dla szyfrowanej liczby – jest ona mnożona przez 2, co powoduje zgodnie z definicją symbolu Jacobi jego zmianę na wartość przeciwną. Wyliczany zawsze jest tylko jeden pierwiastek – r_1. W ten sposób opracowany kryptosystem unika ataków ze znanym szyfrogramem. Odpowiednie spreparowanie końcowych (najmniej znaczących) 4 bitów wiadomości szyfrowanej pozwala wykryć kiedy liczba szyfrowana była mnożona przez 2, aby odtworzyć poprawną wiadomość jednoetapowo, bez zgadywania, który z czterech pierwiastków jest rozwiązaniem.

RAES – połączenie szyfrowania AES-256-CBC oraz algorytmu Rabina. Najpierw następuje generowanie klucza algorytmu AES-256 w sposób losowy. Następnie szyfrowana nim jest wiadomość i do klucza doklejana jest wiadomość zaszyfrowana algorytmem AES-256-CBC (klucz pozostaje jawny na tym etapie). Następnie pierwsze 767 bajtów wiadomości jest szyfrowane algorytmem Rabina ukrywając klucz i początkowe bloki algorytmu AES-256-CBC (wynik szyfrowania ma 768 bajtów). Wynik jest niezdefiniowany dla wiadomości krótszych niż 735 bajtów. Wiadomości takie nie występują w implementacji, ponieważ każda wiadomość poddana algorytmowi Rabina jest wcześniej podpisywana. Toteż informacja jak zachowuje się algorytm w takim przypadku jest irrelewantna.

Podpis kryptograficzny Rabina – najpierw generowanych jest 704 bajtów losowych i są one oznaczane u[0..703]. Pierwszy z tych bajtów musi mieć wartość w przedziale 0–127, aby nie przekroczyć wartością bezwzględną klucza. Następnie wartość «u» jest dopisywana do końca wiadomości podpisywanej i generowany jest skrót SHA-512 według wzoru sha512(m||u), gdzie pionowe kreski oznaczają konkatenację. Następnie komponowane jest 768-bajtowe pole według algorytmu u[0] || SHA512 || u[1..703]. Na tak powstałej liczbie zapisanej w notacji big endian wykonywany jest podpis, czyli wyliczenie pierwiastka r_1. W przypadku, gdy symbol Jacobi względem którejkolwiek z liczb pierwszych stanowiących klucz prywatny jest różny od 1, należy wybrać inną liczbą losową «u» i wszystkie kroki powtórzyć. Ilość prób w celu podpisania średnio wynosi 4.

2.2. Uwierzytelnienie

Pierwszą ramką jaką serwer wysyła, po pełnym połączeniu się, na UDP/DTLS/SCTP/DCEP (wraz z akceptacją nazwy kanału DCEP) jest 256-bitowa sekwencja losowa, którą klient musi podpisać cyfrowo i zaszyfrować odsyłając do serwera. Losowa wartość przesłana do podpisania klientowi uniemożliwia podszycie się pod innego klienta. Wartość ta zabezpiecza także przed atakiem z powtórzoną pierwszą wiadomością.

Budowa ramki żądania uwierzytelnienia:

- 32 losowe bajty.

Odpowiedź na ramkę żądania uwierzytelnienia zawiera trzy argumenty. Jeden o długości 1681 bajtów, drugi do znaku ASCII o kodzie 0 (NUL) oraz trzeci do końca pola danych.

Pierwszy argument (przed zaszyfrowaniem 1632 bajtów, po zaszyfrowaniu 1681 bajtów) zawiera zaszyfrowane algorytmem RAES następujące dane:

- 768 bajtów sygnatury algorytmu Rabin z wiadomości złożonej z poniższych pól:
- 768-bajtowy klucz publiczny (identyfikujący klienta) algorytmu Rabina-6144 (liczba 6144-bitowa) zapisana w notacji big endian,
- echo 32-bajtowej wiadomości od serwera,
- 32-bajtowy klucz szyfrujący algorytmu AES-256,
- nowa 32-bajtowa wartość losowa zwana na potrzeby tej dokumentacji **ciasteczkiem**.

Drugim argumentem jest niezaszyfrowana stała treść zgody na warunki zawarte w niniejszym protokole w dowolnej wydanej wersji zakończona znakiem o kodzie ASCII 0. Treść zgody ma jedną z poniższych postaci:

- «Implementując protokół komunikacyjny dobrowolnie zgadzam się ze wszystkimi zapisami umownymi (w tym dotyczącymi kar umownych i opłat) zawartymi w opisie protokołu Valkyria DMTP, w szczególności potwierdzam, że statusy połączenia zawarte pod flagami 0x04, 0x08 oraz 0x10 działają zgodnie ze specyfikacją opisu protokołu o skrócie SHA-256: [hash].» lub
- «By implementing the communication protocol, I voluntarily agree to all contractual provisions (including those regarding contractual penalties and fees) contained in the Valkyria DMTP protocol description, in particular I confirm that the connection statuses contained under flags 0x04, 0x08 and 0x10 operate in accordance with the protocol description specification with the SHA-256 hash: [hash].».

Zamiast [hash] należy wpisać 64 cyfry szesnastkowe (używając jako cyfr: 0-9, a-f) stanowiące skrót SHA-256 z oryginalnego dokumentu z opisem protokołu dostarczonym na stronie Valkyria.Pro. Zgoda ta w przypadku pracowników wykonujących pracę dla firm – wyrażana jest w imieniu pracodawcy / firmy zlecającej / ostatniego znanego licencjodawcy na oprogramowanie, w którym nastąpiła implementacja niniejszego protokołu (niezależnie czy klienckiego, czy serwerowego).

Trzeci argument (do końca pola danych protokołu transportowego) zawiera zaszyfrowaną algorytmem AES-256-CBC (z wektorem inicjującym zerowym) treść umowy dotyczącej poufności użytkowników oprogramowania klienckiego zakodowaną w formacie UTF-8, w języku angielskim lub polskim, która musi być zgodna z prawdą i pełna. Kluczem szyfrującym jest ustalony wyżej klucz.

W przypadku, gdy serwer odbierze niepoprawną odpowiedź uwierzytelniającą – nie odsyła odpowiedzi. Takie połączenie podlega przerwaniu.

W przypadku, gdy serwer odbierze nieprawidłową treść zgody – nie odsyła odpowiedzi.

W przypadku, gdy odszyfrowana treść umowy o prywatności oprogramowania klienta jest nieczytelna lub źle sformatowana w UTF-8, serwer powinien zerwać połączenie. Nieakceptowalna umowa o prywatność (ale czytelna) nie powinna kończyć połączenia. Decyzję o jej akceptacji powinien wyrazić użytkownik końcowy.

W przypadku, gdy serwer odbierze poprawną odpowiedź uwierzytelniającą, poprawną zgodę i czytelną umowę o prywatność aplikacji klienckiej, serwer odsyła ciasteczko zaszyfrowane uzgodnionym kluczem szyfrującym AES-256 w trybie CBC z IV=0.

Klient, do którego nie wróci poprawna zaszyfrowana wiadomość (zawierająca 48 bajtów po zaszyfrowaniu) zawierająca ciasteczko, powinien się rozłączyć. Brak powrotu ciasteczka świadczy o tym, że ktoś jest między nadawcą a odbiorcą lub też, że nastąpił inny nieprzewidziany błąd.

2.2.1. Treść umowy o poufność oraz konsekwencje jej zatajania

W sytuacji, gdy klient przysyła treść umowy niezapewniającą odpowiedniego stopnia poufności danych użytkownika, użytkownik taki powinien mieć w statusie połączenia zwracanym przez serwer ustawiony bit 0x04 oznaczający brak spełnienia zasad poufności i bezpieczeństwa. Gdy nie jest możliwe stwierdzenie, czy dane użytkownika są odpowiednio chronione, należy ustawić bit 0x08. Akceptowalność umów musi sprawdzać człowiek, a na bazie skrótów SHA-256 powtórnego sprawdzenia może dokonywać maszyna.

Programista (w tym firma, organizacja, inna osoba prawna) podejmująca się implementacji niniejszego protokołu, która spowoduje brak ustawiania bitu 0x04 lub 0x08 (oznaczającego nieakceptowalną lub nieznaną politykę prywatności i tym samym powodując brak wyświetlenia okienka z umową o poufność u użytkownika) przyjmuje dobrowolnie na siebie ciężar zapłaty odszkodowań (nie mniejszych niż cena połowy uncji złota każde) na rzecz osoby, która nie będzie odpowiednio poinformowana o polityce prywatności oprogramowania klienckiego oraz na rzecz autorki niniejszego protokołu. Użycie w statusie bitu 0x08 zamiast 0x04 (lub odwrotnie) nie podlega odszkodowaniom, bowiem użytkownik końcowy (klient) będzie miał wyświetloną treść umowy.

Wysyłanie przez klienta celowo umowy zawierającej inną treść niż ma to miejsce w rzeczywistości (czyli kłamstwo na temat przetwarzania danych osobowych) na podstawie treści niniejszego opisu protokołu komunikacyjnego podlega odszkodowaniom (nie mniejszych niż cena połowy uncji złota każde) płatnym dla autorki niniejszego protokołu, osoby zgłaszającej takie naruszenie oraz dla osoby udostępniającej oszukany przez aplikację serwer. Odszkodowania te mają za cel zniechęcenie do łamania praw Europejczyków do prywatności.

Uprawnione jest nieustawienie bitu 0x04 i 0x08 (jednocześnie) w statusie połączenia, gdy aplikacja nie przesyła nikomu statystyk ani treści rozmów i nie umożliwia wglądu w nie trzecim podmiotom. Gdy aplikacja kliencka jest skompilowana na system operacyjny znany z łamania praw człowieka (np. Microsoft Windows, Apple iOS, MacOS, Google Android) musi umieścić o tym informację w treści umowy o prywatność.

Serwery działające pod kontrolą systemów operacyjnych znanych z łamania praw człowieka muszą ustawiać bit 0x10 statusu połączenia. Dotyczy to szczególnie serwerów działających pod kontrolą

Microsoft Windows, Apple iOS, MacOS, Google Android oraz serwerów umieszczonych w chmurach Amazona, Azure oraz GPC.

Niezastosowanie przez serwer statusu połączenia z włączoną flagą 0x10 jest przedmiotem odszkodowań (każde nie mniejsze niż połowa ceny uncji złota) dla autorki protokołu, osoby zgłaszającej incydent płatnych z ręki osoby uruchamiającej taki serwer. Flagi połączeń są szczegółowo opisane w rozdziale «odczyt statusu użytkownika».

Powyższe odszkodowania mogą być dochodzone w trybie przepisów o odszkodowanie, z tytułu opłaty licencyjnej, praw autorskich lub też jako kara umowna, bowiem zapoznanie się z treścią niniejszego protokołu jest jednoznaczne z zawarciem umowy. Jeżeli nie zgadzasz się na zawarcie umowy – nie korzystaj z informacji zawartych w tym dokumencie ani tej wiedzy nie przekazuj dalej.

2.2.2. Bezpieczeństwo i poufność

Zastosowanie losowej wartości powitalnej od serwera zabezpiecza przed atakami polegającymi na powtórzeniu szyfrogramów wcześniej nagranych.

Zastosowanie klucza publicznego algorytmu Rabin-6144 uniemożliwia podszywanie się pod danego klienta innej osoby, bowiem inny klient nie zna klucza prywatnego potrzebnego do wyliczenia sygnatury (podpisu). Poprawna sygnatura oznacza, że klient jest rzeczywiście tym klientem, za którego się podaje.

Zastosowanie szyfrowania asymetrycznego w kierunku serwera uniemożliwia zapoznanie się z treścią wiadomości wysyłanej do serwera, zawierającej klucz AES-256 stosowany do szyfrowania pozostałych komunikatów. Klucz ten zna tylko serwer i klient i nikt po środku. Do jego odszyfrowania bowiem jest potrzebny klucz prywatny serwera, który zna tylko serwer.

Odesłanie ciasteczka podpisanego ustalonym kluczem pozwala dowiedzieć się klientowi, że serwer poprawnie zrozumiał jaki ustalono klucz szyfrujący do dalszej komunikacji.

2.3. Wymiana danych

Od razu po uwierzytelnieniu i wymianie klucza szyfrującego AES-256 możliwa jest wymiana danych.

Początkowy wektor inicjujący klienta (o długości 16 bajtów) jest równy pierwszym 16 bajtom ciasteczka przesyłanego na etapie uwierzytelnienia. Początkowy wektor inicjujący serwera jest równy ostatnim szesnastu bajtom tego ciasteczka. Wektor ten jest zwiększany o jeden wraz z każdą kolejną wysłaną wiadomością modulo 2^{128}.

Każda wiadomość (poza danymi uwierzytelniającymi) składa się z jawnie podanych 4 ostatnich (najmniej znaczących) bajtów wektora inicjującego oraz zaszyfrowanego **komunikatu**.

Każdy komunikat składa się ze:

- skrótu SHA-256, którym zabezpieczone są poniższe informacje,
- 3-bajtowego numeru porządkowego komunikatu – zaczynając od zera po uwierzytelnieniu,
- 1-bajtowego typu komunikatu (opisanego w następnym rozdziale),
- danych (atrybutów) o różnej długości charakterystycznych dla danego komunikatu.

3. Komunikaty protokołu DMTP

3.1. Budowa każdego komunikatu

Pierwsze 32 bajty komunikatu stanowi suma kontrolna policzona algorytmem SHA-256. Suma ta zabezpiecza integralność komunikatu na wypadek prób fałszowania wartości wektora inicjującego poprzez wycinanie lub powtarzanie wiadomości. Komunikat taki po odszyfrowaniu będzie miał sumę kontrolną niepasującą do reszty zawartości i zostanie odrzucony.

Po sumie występuje 3-bajtowy kolejny numer komunikatu. Numer komunikatu pozwala powiązać asynchroniczną odpowiedź z poleceniem. Żądania i odpowiedzi mają bowiem ten sam numer. Zalecane jest stosowanie kolejnych numerów dla kolejnych komunikatów, co zabezpiecza przed wczesną powtórką numeru i błędnym powiązaniem żądania i odpowiedzi na nie.

Po numerze polecenia znajduje się jego typ (kod) zapisany jednobajtowo. Żądania mają kody od 0x01 do 0x7E, zaś odpowiedzi na te żądania mają odpowiednio kody 0x81 do 0xFE.

Wartość 0x80 zarezerwowana jest dla powiadomień o nowych wiadomościach na serwerze. Wartości 0x00, 0x7F oraz 0xFF są zarezerwowane do przyszłych zastosowań. Nie należy ich używać.

W rozdziałach poniższych opisane są numery typów komunikatów wraz ze znaczeniem danych, które są przesyłane wraz z tym numerem.

Jeżeli nie podano w opisie polecenia inaczej – wartości liczbowe zapisywane są w notacji big endian.

3.2. Indeksy

Aby rozróżniać w systemie użytkowników zostały wprowadzone indeksy. Indeks jest ciągiem 32 bajtów. Każdy indeks składa się z 256 bitów zapisanych jako 32 bajty.

Pierwsze cztery bajty stanowią numer porządkowy użytkownika. Numer porządkowy pozwala szybko odnajdować pozycję w plikach roboczych serwera.

Kolejne 27 bajtów to liczba losowa. Liczba ta zapobiega wysyłaniu niechcianych wiadomości do użytkowników. Gdyby liczby tej nie wydłużyć do 256 bitów możliwe byłoby zgadywanie numerów użytkowników bez znajomości ich kluczy oraz zakłócanie transmisji bezsensownymi wiadomościami (nie dającymi się odszyfrować u odbiorcy).

Kolejny 1 bajt stanowi informacje statusowe. To pole jest istotne tylko gdy jest odbierane z serwera. Podczas wysyłania jest ignorowane. Zaleca się by podczas wydawania poleceń było ono równe zero.

Indeksy są stosowane zamiast kluczy publicznych dlatego, że klucze publiczne tożsamości kryptograficznej zajmują 768 bajtów. To stanowczo za dużo, aby sprawnie nadawać wiadomości. Przechowywanie wszędzie kluczy przy każdej wiadomości marnotrawiłoby ponadto zasoby serwera, a odszukiwanie po kluczu ma złożoność wyższą niż po indeksie, który zawiera liczbą porządkową.

3.3. Odpowiedzi z błędem

W przypadku, gdy wystąpi błąd podczas przetwarzania polecenia i błąd ten nie jest błędem krytycznym, odpowiedź z błędem jest zwracana jak zwykła wiadomość (odpowiedź z kodem 0x81 do 0xFE) i posiada ona długość jednego bajta danych. Kody błędów wyszczególnione są pod poleceniami, których dotyczą.

W przypadku błędu krytycznego odpowiedź nie jest w ogóle odsyłana.

3.4. Odczyt statystyk serwera

Polecenie ma numer 0x1E. Odpowiedź na to polecenie ma kod 0x9E. Polecenie to jako parametr przyjmuje jeden bajt wskazujący rodzaj danych oczekiwanych przez klienta.

3.4.1. Odczyt czasu serwera i rozmiaru bufora

Jako parametr dla odczytu statystyk serwera należy wysłać bajt 0x00. Serwer zwraca aktualny czas w milisekundach po północy dnia 1970-01-01 UTC (6 bajtów) oraz maksymalny rozmiar bufora na dane – 4 bajty.

3.4.2. Odczyt danych kontrolnych protokołu DMTP

Jako parametr należy podać wartość 0x02. Serwer zwraca cztery 64-bitowe wartości:

- ilość odebranych poprawnych poleceń,
- ilość nieprawidłowo zakodowanych poleceń (niepoprawnie zaszyfrowane lub zła suma kontrolna),
- ilość zagubionych poleceń (wektor inicjujący różni się o więcej niż 1, ale mniej niż 255),
- czas od startu serwera w milisekundach – pozwala wykryć restarty serwera,
- ilość błędów krytycznych operacji wejścia/wyjścia.

3.4.3. Odczyt ilości zużywanej pamięci

Jako parametr dla odczytu statystyk serwera należy podać 0x01. Serwer zwraca 2 liczby 32-bitowe:

- ilość aktualnie zajmowanej pamięci wirtualnej w kilobajtach,
- ilość zajmowanej pamięci wirtualnej w szczycie w kilobajtach (maximum jakie wystąpiło od chwili uruchomienia serwera),
- maksymalna dopuszczalna wielkość skrzynki odbiorczej w bajtach.

3.5. Odczyt klucza publicznego

Polecenie ma numer 0x02. Odpowiedź na to polecenie ma kod 0x82. Polecenie to przyjmuje jeden parametr: 32-bajtowy indeks identyfikujący użytkownika.

Polecenie to zwraca 768 bajtów, które reprezentują klucz publiczny Rabin-6144 zapisany w notacji big endian. Klucz ten jednoznacznie identyfikuje użytkownika i pozwala sprawdzić jego tożsamość.

Kody błędów:

- 0x01 – długość parametru jest nieprawidłowa.
- 0x02 – zadana wartość parametru nie przedstawia żadnego użytkownika na serwerze.

3.6. Odczyt statusu użytkownika

Należy odróżnić status przesyłany wewnętrznie przez klienta Valkyria Besiada (porozmawiajmy, zaraz wracam, nie mam czasu) od statusu na serwerze. Status na serwerze dotyczy połączenia.

Polecenie ma numer 0x03. Odpowiedź na to polecenie ma kod 0x83. Jako dane wejściowe można podać od 0 do 768 indeksów identyfikujących użytkownika. Jako odpowiedź wróci tyle bajtów odpowiedzi, ile wprowadzono indeksów. Zwrócona wartość 0xFF oznacza błąd – np. brak identyfikacji danego użytkownika.

Bajty danych odpowiedzi są polami flagowymi. Poszczególne bity (flagi) mają znaczenie następujące:

- 0x01 – adres publiczny IP użytkownika jest inny niż klienta pytającego,
- 0x02 – wskazany użytkownik jest online lub był online w przeciągu 3 ostatnich minut,
- 0x04 – treść umowy o prywatność klienta zawiera ustalenia dotyczące przetwarzania danych użytkownika do innych celów niż samo dostarczenie wiadomości, w szczególności klient może być uruchomiony na systemie operacyjnym Microsoft Windows, Apple iOS, Apple MacOS, Google Android,
- 0x08 – treść umowy o prywatność klienta jest niezweryfikowana przez człowieka w zakresie prywatności,
- 0x10 – serwer jest uruchomiony na systemie operacyjnym znanym z łamania praw człowieka, w szczególności na Microsoft Windows, Apple iOS, Apple MacOS, Google Android.

Zwrócenie flag 0x04 lub 0x08 (lub obu jednocześnie) musi spowodować w kliencie zapytanie o umowę prywatności i jej zapamiętanie w kliencie oraz wyświetlenie jej na każde żądanie użytkownika. Prezentowana powinna być też czerwona otwarta kłódka obok opisu użytkownika.

Zwrócenie flagi 0x10 powinno spowodować tylko wyświetlenie informacji o możliwości śledzenia kontaktów przez prywatne podmioty trzecie. Odczytywanie treści komunikacji jest bowiem nadal niemożliwe.

W przypadku błędnej składni danych polecenia – zwracana jest pusta odpowiedź.

Uwaga: w przypadku posiadania znajomych więcej niż 768, należy wysłać kilka oddzielnych zapytań, nie przekraczając maksymalnej ilości indeksów w jednym poleceniu. Odpowiedź na dłuższe zapytania nie jest bowiem gwarantowana.

3.7. Odczyt umowy oprogramowania klienckiego

Oprócz umowy licencyjnej i polityki prywatności, które akceptujemy w używanym przez nas kliencie, istotne jest wiedzieć jakie warunki prywatności przewiduje oprogramowanie po drugiej stronie, bowiem tam są wyświetlane prywatne wiadomości, które piszemy. Nie można więc usprawiedliwić tajemnicy tego, jakie warunki prywatności przewiduje oprogramowanie po drugiej stronie łącza. Ważnym bowiem jest to, kto ma dostęp do wiadomości wysyłanych do konkretnej osoby.

Polecenie to ma numer 0x01 i przyjmuje jeden argument, którym jest 32-bajtowy indeks użytkownika. Odpowiedź na to polecenie ma kod 0x81.

Polecenie to zwraca jako odpowiedź treść umowy o prywatność, jaką zadeklarował klient podczas łączenia się.

Polecenie to może zwrócić błędy:

- 0x01 – niepoprawna długość argumentu,
- 0x02 – nie znaleziono użytkownika o wskazanym indeksie.

3.8. Odczyt indeksów użytkowników

Polecenie ma numer 0x04. Odpowiedź na to polecenie ma kod 0x84. Jako dane wejściowe można podać od 0 do 32 kluczy tożsamości kryptograficznej użytkowników. Jako odpowiedź wróci tyle indeksów, ile wprowadzono kluczy. Zwrócony indeks zawierający 32 bajty 0xFF oznacza, że użytkownik w niedawnej przeszłości (do 30 dni) nie używał klienta zgodnego z DMTP-2.01 lub też usunął informację o kluczu publicznym.

Możliwe błędy:

- 0x01 – zapytanie ma błędny rozmiar,
- 0x02 – za dużo kluczy w jednym zapytaniu.

Ostatni bajt każdego indeksu w tej odpowiedzi jest statusem połączenia, o którym mowa jest w poprzednim podrozdziale. Nie ma więc potrzeby po odczytaniu indeksów pytać ponownie o status.

Uwaga: w przypadku posiadania znajomych więcej niż 32, należy wysłać kilka oddzielnych zapytań, nie przekraczając 32 kluczy 768-bajtowych w jednym zapytaniu. Odpowiedź na dłuższe zapytania nie jest gwarantowana.

3.9. Konwersja skrótu na klucz publiczny

Od wersji Valkyria Besiada 1.06 możliwe jest zapraszanie użytkowników za pomocą skrótu SHA-256 z ich klucza publicznego. Polecenie konwersji skrótu na klucz publiczny pozwala odnaleźć w pamięci serwera klucz publiczny zwracający zadany skrót SHA-256.

Polecenie ma numer 0x08. Odpowiedź na to polecenie ma kod 0x88.

Danymi wejściowymi jest skrót SHA-256 zapisany binarnie (32 bajty), zaś odpowiedzią jest 768-bajtowy klucz publiczny.

Możliwe błędy:

- 0x01 – błędna długość zapytania,
- 0x02 – klucza nie znaleziono.

3.10. Wysłanie wiadomości

Główną funkcją serwera jest umożliwienie przesyłania wiadomości i to polecenie realizuje tą funkcję poprzez dostarczenie zaszyfrowanej wcześniej wiadomości do jednego klienta.

Polecenie ma numer 0x05. Odpowiedź na to polecenie ma kod 0x85.

Polecenie to przyjmuje dwa argumenty: 32-bajtowy indeks adresata oraz zaszyfrowaną treść wiadomości. Zaszyfrowana treść wiadomości musi mieć długość wielokrotności 16 bajtów.

Odpowiedź na to polecenie może przyjmować następujące jednobajtowe wartości:

- 0x00 – wiadomość udało się umieścić w skrzynce odbiorczej u adresata,
- 0x01 – nieprawidłowo zbudowane żądanie,
- 0x02 – nieznany adresat,
- 0x03 – błąd sprzętowy podczas zapisu wiadomości,
- 0x04 – adresat nie ma skrzynki odbiorczej na tym serwerze,
- 0x05 – przepełnienie skrzynki odbiorczej adresata.

Wysłanie tej wiadomości może powodować wystąpienie powiadomień u adresata, gdy jest on zalogowany i włączył dostarczanie asynchroniczne powiadomień.

3.11. Kasowanie wiadomości

Aby nie ginęły wiadomości, wymagane jest wysłanie tego polecenia do skasowania wiadomości. Wiadomość będzie się znajdować w skrzynce odbiorczej do czasu wysłania polecenia skasowania.

Polecenie ma numer 0x07. Odpowiedź na to polecenie ma kod 0x87.

Argumentem dla tego polecenia jest dodatnia liczba 32-bitowa wskazująca ilość bajtów do skasowania.

Odpowiedź na to polecenie jest jednobajtowa i przyjmować może wartości:

- 0x00 – udało się prawidłowo wykonać polecenie,
- 0x02 – w zadanym miejscu nie można skasować, ponieważ nie przypada w tym miejscu podział wiadomości,
- 0x03 – wskazano za daleką pozycję do skasowania, ponieważ w buforze nie ma tyle treści,
- 0x04 – zawartość skrzynki jest uszkodzona i wymagane jest skasowanie całości,
- 0x05 – wystąpił nieznany wyjątek podczas próby kasowania wiadomości.

Polecenie może przyjmować dwie szczególne wartości opisane niżej.

3.11.1. Włączenie powiadomień asynchronicznych

Aby móc odbierać wiadomości niezbędne jest włączenie powiadomień asynchronicznych. Polecenie to należy stosować tylko na głównym serwerze komunikacyjnym.

Aby włączyć powiadomienia asynchroniczne należy wysłać polecenie kasowania wiadomości z atrybutem w postaci zera zapisanego na 32 bitach. Odpowiedź na to polecenie może przyjąć wszystkie powyższe wartości poza 0x02.

Polecenie to jednocześnie zakłada skrzynkę odbiorczą, gdy nie była założona.

3.11.2. Skasowanie skrzynki odbiorczej

Aby móc się komunikować na innych serwerach niż główny, trzeba się w nich zarejestrować. Rejestracja powoduje jednocześnie założenie skrzynki. Aby skasować całą skrzynkę odbiorczą, a nie tylko wiadomości, należy wysłać polecenie kasowania wiadomości z wartością 0xFFFFFFFF jako argument zapisaną na 32 bitach.

Polecenie to jest przydatne także w sytuacji wystąpienia błędu 0x04 podczas próby kasowania wiadomości. Ponowne włączenie powiadomień asynchronicznych spowoduje utworzenie pustej skrzynki odbiorczej.

Od tej chwili próby wysłania wiadomości na danym serwerze ze wskazanym indeksem odbiorcy zakończą się niepowodzeniem i serwer nie przydziela przestrzeni dyskowej dla użytkownika.

Odpowiedź na to polecenie powinna być jednobajtowa i wynosić zawsze 0x00. Inna wartość wskazuje na błąd systemu operacyjnego lub błąd sprzętowy.

3.12. Wysłanie wiadomości grupowej

Serwer umożliwia wysłanie wiadomości grupowej do wielu odbiorców, potocznie nazywanej multikastem. Ponieważ zaszyfrowana wcześniej wiadomość wygląda inaczej podczas wysyłki do każdego z odbiorców, a ma jedynie tą samą długość, nie jest to multikast w pełnym rozumieniu tego słowa. Powoduje to, że wiadomość wciąż musi zawierać pełną listę odbiorców oraz powtórzony szyfrogram wiadomości dla każdego z odbiorców. Umożliwia to jednak wysłanie wiadomości statusowych (klucze, opisy, statusy programu Valkyria Besiada) jednocześnie do wszystkich wyspecyfikowanych adresatów.

Procedura taka jest zalecana na przykład podczas rozsyłania informacji o przejściu w tryb offline, kiedy trzeba tą informację wysłać szybko ze względu na ograniczenia nałożone na przeglądarki (powiadomienie o zamknięciu okna lub zakładki).

Polecenie to posiada numer 0x06. Odpowiedź na to polecenie ma kod 0x86.

Jako pierwszy jednobajtowy argument należy podać długość wiadomości wysyłanej do każdego odbiorcy podzieloną przez 16. Oznacza to, że najdłuższa wiadomość wysyłana za pomocą tego polecenia może mieć długość 4080 bajtów.

Drugim argumentem jest lista zawierająca przynajmniej jeden element, składający się z poniższych pól:

- 32-bajtowy indeks odbiorcy,
- zaszyfrowana wiadomość dla danego odbiorcy.

Odpowiedź na to polecenie zawiera listę kodów błędów, które wystąpiły podczas obsługi zapytania. Długość listy kodów błędów wynosi tyle, ile adresatów wskazano podczas wysyłania polecenia.

Kody błędów:

- 0x00 – wiadomość poprawnie dostarczono do adresata,
- 0x01 – błędny format polecenia (zwracany jest w tym przypadku tylko jeden bajt),
- 0x02 – nieznany adresat,

- 0x03 – błąd sprzętowy podczas zapisu wiadomości,
- 0x04 – brak skrzynki odbiorczej u adresata na tym serwerze,
- 0x05 – przepełnienie skrzynki odbiorczej u adresata.

Długość łączna argumentu nie może przekraczać 32000 bajtów. Jeżeli potrzeba jest wysłać więcej wiadomości, należy to wykonać poprzez wysłanie kilku kolejnych poleceń, bowiem przyjęcie polecenia zawierającego argument dłuższy niż 32000 bajtów nie jest gwarantowane.

3.13. Echo

Polecenie to służy do celów diagnostycznych sprawdzania połączenia. Umożliwia wygenerowanie pakietów o różnej długości. Polecenie to ma kod 0x0E i jako parametr przyjmuje dowolny ciąg binarny o dowolnej długości od 0 do 64000 bajtów. Odpowiedź ma kod 0x8E i zawiera powtórzony z żądania ciąg jako odpowiedź.

3.14. Usunięcie konta

Ze względu na fakt, że klucz tożsamości kryptograficznej identyfikuje użytkownika oraz na serwerze przechowywany jest skrót MD5 z adresu IP (MD5 liczone celem ukrycia IPv6, nie do celów kryptograficznych) wprowadzona została komenda usunięcia tych danych dla pełnej zgodności z RODO. Wydanie tej komendy usunie powyższe dane, jednak nie usunie wysłanych wiadomości. Indeks zawarty w wiadomościach nie będzie jednak już dalej rozpoznawany jako poprawny. Po wysłaniu tej komendy i uzyskaniu odpowiedzi – dalsza komunikacja nie będzie już możliwa.

Polecenie to ma kod 0x0D. Odpowiedź na to polecenie ma kod 0x8D.

Argumentem dla polecenia jest klucz własnej tożsamości kryptograficznej (768 bajtów) służący jako potwierdzenie chęci usunięcia konta.

Odpowiedzią dla tego polecenia jest jednobajtowy kod błędu:

- 0x00 – poprawnie usunięto konto wraz ze skrzynką odbiorczą,
- 0x01 – niepoprawny format argumentu,
- 0x02 – odmowa usunięcia z powodu podania nieswojego klucza tożsamości kryptograficznej,
- 0x03 – błąd sprzętowy (lub systemu operacyjnego) podczas operacji usuwania danych.

Po wydaniu tego polecenia kolejne polecenia, do czasu kolejnego uwierzytelnienia, pozostają bez odpowiedzi, jednak klient pozostaje połączony. Zachowanie takie ma na celu zapobieżenie ponownemu połączeniu się i dodaniu klucza tożsamości kryptograficznej do spisu.

Po wydaniu tego polecenia należy się niezwłocznie rozłączyć.

Uwaga: w programie Valkyria Besiada klucz własnej tożsamości kryptograficznej, potrzebny jako argument tego polecenia, odczytuje się poleceniem «/mój-klucz».

3.15. Aktualizacja oprogramowania

DMTP umożliwia aktualizację oprogramowania. Celem zapewnienia autentyczności oprogramowanie powinno być podpisywane cyfrowo przez osoby je tworzące. Transfer oprogramowania odbywa

się fragmentami o rozmiarze do 64000 bajtów. Obraz pliku oprogramowania składa się z ciągłej przestrzeni obejmującej według kolejności:

- 32-bitowy rozmiar pliku obejmujący również ten nagłówek,
- 64-bajtową sumę kontrolną SHA-512,
- dane binarne pliku do pobrania.

Polecenie do pobrania oprogramowania ma kod 0x0C. Odpowiedź na to polecenie ma kod 0x8C. Polecenie to przyjmuje następujące parametry:

- 32-bitowy offset względem początku nagłówka,
- 16-bitowy rozmiar żądanego bloku,
- wersję oprogramowania zakończoną znakiem NUL (dostępna na ten moment tylko wersja «html»),
- lista dwuliterowych kodów języków według ISO-639-1 (każdy zapisany na 2 bajtach).

Odpowiedź na żądanie zawiera fragment obrazu oprogramowania. Powrót mniejszej ilości bajtów niż zażądano oznacza koniec pliku.

Po pobraniu pliku koniecznie trzeba policzyć sumę kontrolną, która zabezpiecza przed niespójnością spowodowaną podmianą obrazu pliku w trakcie pobierania.

Oprogramowanie może być dostarczane w spakowanym archiwum gz – może wymagać dekompresji. Suma kontrolna w takim przypadku dotyczy stanu skompresowanego.

3.16. Zmiana klucza szyfrującego AES-256

Klient po osiągnięciu pewnego limitu wiadomości może wysłać do serwera żądanie zmiany klucza ustalonego na etapie uwierzytelniania klucza szyfrującego. Zaleca się, zmianę co każde 100 MB danych (łącznie w obie strony) klucza szyfrującego, aby utrudnić odgadnięcie klucza po cechach charakterystycznych szyfru i wysyłanych wiadomości.

Polecenie do zmiany klucza szyfrującego ma kod 0x0F i przyjmuje argument 32-bajtowy, będący nowym kluczem szyfrującym algorytmu AES-256. Odpowiedź na to polecenie ma kod 0x8F.

Odpowiedź na poprawnie sformatowane polecenie zawsze zawiera jeden bajt: 0x00. Odpowiedź na błędnie sformatowane polecenie zawiera jeden bajt 0x01.

Uwaga: wektory inicjujące klienta i serwera pozostają niezmienne. Odpowiedź na to polecenie jest zaszyfrowana nowym kluczem.

3.17. Powiadomienie z odebranymi wiadomościami

Po włączeniu odbioru powiadomień asynchronicznych (patrz: kasowanie wiadomości, wartości szczególne) asynchronicznie są przysyłane powiadomienia o nowych wiadomościach.

Powiadomienie takie przychodzi w dowolnym momencie od serwera i ma wartość polecenia 0x80. Jako numer polecenia może zawiera 24-bitowe pole flag, dla którego kolejne flagi mają następujące znaczenie:

- 0x000001 – poza zawartymi w odpowiedzi wiadomościami nie ma więcej wiadomości w skrzynce odbiorczej,
- 0x000FFE – flagi zarezerwowane do przyszłego użycia, w przypadku odebrania wiadomości powinny być ignorowane i obsłużone tak, jakby ich nie było,
- 0xFFF000 – flagi zarezerwowane do przyszłego użycia, obecność dowolnej powinna skutkować odrzuceniem komunikatu (znaczenie odpowiedzi jest znacznie zmienione).

Pole komunikatu zawiera listę wiadomości zmiennej długości o poniższej budowie:

- 32-bitowa długość obejmująca poniższe elementy (ale nie samą siebie),
- 32-bajtowy indeks nadawcy na serwerze, z którego pochodzi wiadomość,
- zaszyfrowana wiadomość od nadawcy o długości stanowiącej wielokrotność 16 bajtów.

Długość powiadomienia nie może przekraczać 64000 bajtów. Wiadomości należy skasować (polecenie kasowania wiadomości) wskazując jako parametr całkowitą długość wiadomości zawartą w niniejszym komunikacie.

3.18. Spis wszystkich kodów poleceń

0x01 – odczyt umowy oprogramowania klienckiego,

0x02 – odczyt klucza publicznego,

0x03 – odczyt statusu użytkownika,

0x04 – odczyt indeksów użytkowników,

0x05 – wysłanie wiadomości,

0x06 – wysłanie wiadomości grupowej,

0x07 – kasowanie wiadomości,

0x08 – konwersja skrótu na klucz publiczny,

0x0D – usunięcie konta,

0x0E – echo,

0x0F – zmiana klucza szyfrującego AES-256,

0x1E – odczyt statystyk serwera,

0x80 – powiadomienie z odebranymi wiadomościami.

4. Uwagi końcowe

Protokół niniejszy celowo stworzono tak prostym jak to jest możliwe. Z tego powodu nie ma w nim XML ani JSON polepszającego czytelność danych. Celem jest szybka obsługa na serwerach, konsumpcja tak małej ilości zasobów, jak to jest możliwe oraz możliwość łatwego zapoznania się z właściwościami protokołu przez użytkowników chcących sprawdzić bezpieczeństwo niniejszej implementacji.

4.1. Krytyczne uwagi dotyczące bezpieczeństwa

Ponieważ możliwy jest atak ze znanymi dwoma szyfrogramami na kryptosystem Rabina, gdy zaszyfrowaną wiadomość stanowią dwa różne pierwiastki (np. r_1 i r_3), odradza się dla bezpieczeństwa właściciela klucza, używania go w innych aplikacjach. W sytuacji, gdy dwie aplikacje będą używały innych wzorów do wyliczenia pierwiastka (np. r_1 i r_3), kompromitacji ulega klucz, który został użyty.

Jednym ze sposobów uniknięcia problemu używania różnych wzorów do wyliczania oryginalnej wiadomości i innych wzorów do podpisywania przez dwie różne aplikacje jest używanie klucza sprzętowego podłączonego do serwera jako urządzenie na UART, realizujące polecenia podpisz oraz odszyfruj.

Realizacja szczególnie podpisu cyfrowego dwoma różnymi algorytmami liczenia tzw. pierwiastka (r_1 do r_4) oraz realizacja deszyfrowania i przesyłania dalej odszyfrowanej wiadomości różnymi wzorami na pierwiastek, zawsze kończy się krytycznie źle dla właściciela klucza serwerowego!

4.2. Status prawny niniejszego dokumentu

Niniejszy dokument jest dziełem w rozumieniu przepisów o prawie autorskim. Autorka wyraża zgodę na bezpłatne rozpowszechnianie niniejszego dzieła jednak tylko wówczas, gdy dokument ten jest wykorzystywany do tworzenia bezpłatnego oprogramowania, przy czym za płatne wykorzystywanie uważa się także sytuację, gdy użytkownik końcowy płaci danymi (w tym osobowymi).

Firmy prywatne niezainteresowane dostarczaniem oprogramowania, które chroni prywatność użytkownika w stopniu tak wysokim jak to tylko możliwe – przewidziane jest dostarczenie opisu protokołu po wniesieniu opłaty, która zostanie zaproponowana osobie, która chce nabyć ten protokół do celu implementacji w swoich płatnych produktach (w tym płatnych danymi osobowymi). Dokument ten nie będzie jednak nigdy zawierał zezwolenia na okłamywanie (lub zatajanie) polityki prywatności klienta podłączonego po drugiej stronie łącza.

Za łamanie postanowień niniejszego dokumentu przewidziane jest «odszkodowanie», które może być dochodzone jako opłata licencyjna, opłata z tytułu praw autorskich, odszkodowanie w rozumieniu prawa lub kara umowna. Szczegóły znajdują się w podrozdziale «Treść umowy o poufność oraz konsekwencje jej zatajania». Tym samym protokół komunikacyjny Valkyria DMTP-2.01 nie pozostaje tajny i jest przewidziana możliwość interoperacyjności, o której mówi rozporządzenie DMA. Prawo do interoperacyjności jednak nie może łamać innych praw obowiązujących w Europie, w tym prawa do prywatności i ochrony danych osobowych oraz prawa do przejrzystości w zakresie, jak wykorzystywane są dane o Europejczykach.