EXPATCH LLC

VULNERABILITY RESEARCH · COORDINATED DISCLOSURE

# Мы доверили Telegram данные об уязвимости ради пользователей — и нас исключили из истории её исправления

Техническая история NTLM relay, раскрытия heap-памяти и исправлений в Telegram Desktop и QtNetwork.

**Александр Ростилов**
**Денис Ростилов**
ExPatch Vulnerability Research Team

**Сентябрь 2026**
Telegram Desktop 6.8.x · Qt 6.11.1
Boston, Massachusetts

КРАТКОЕ РЕЗЮМЕ

# Что произошло

Исследование ExPatch выявило два независимых последствия в пути аутентификации QtNetwork, достижимом из официального Telegram Desktop: перенаправление NTLM-аутентификации и повторяемое раскрытие памяти процесса.

### Автоматический NTLM

Ответ 407 от контролируемого прокси мог без отдельного согласия пользователя запустить SSPI и сформировать NTLM-ответ текущей Windows-сессии.

### Лабораторная RCE-цепочка

В изолированном домене relay к AD CS приводил к выполнению кода в пределах прав скомпрометированной учётной записи; для Domain Admin подтверждён SYSTEM.

### До 65 КБ за обмен

Ошибка в qExtractServerTime() включала в NTLMv2-ответ до 64 992 байт непрочитанной heap-памяти; 100 циклов раскрыли около 6,5 МБ.

### Вопрос атрибуции

Полный отчёт передан Telegram 6 июня; совпадающие патчи датированы 8 июня; upstream-работа Qt началась 10 июня, а точное heap-исправление оформлено 12 июня. В публичных записях авторы исследования не указаны.

| ОСНОВНОЙ АНАЛИЗ | БИБЛИОТЕКА | RCE-СТЕНД |
|---|---|---|
| **Telegram Desktop 6.8.4** | **Qt 6.11.1 / QtNetwork** | **Telegram Desktop 6.8.2** |

НАВИГАЦИЯ

# Содержание

> «Наш опыт сформирован нашей миссией — защищать наших пользователей в условиях авторитарных режимов».

— Павел Дуров, 5 сентября 2024 года. Перевод ExPatch; оригинал.


**Рис. 1.** *Редакционный фрагмент оригинального поста Павла Дурова: шапка автора, цитируемый абзац и метаданные публикации; пропущенные части обозначены многоточием.*

Мы согласны с этими словами. Более того — мы им поверили.

Для человека, живущего в условиях цензуры, прокси Telegram — не просто пункт в настройках приложения. Иногда это последняя узкая тропа к независимой информации и свободному интернету. Но тропа к свободе превращается в ловушку, если способна незаметно раскрыть учётные данные, содержимое памяти или передать злоумышленнику контроль над устройством.

Обнаружив именно такую опасность, мы могли сначала обратиться в Qt, зафиксировать свой приоритет и ждать обычного цикла исправления. Мы поступили иначе. В ночь перед отправкой отчёта мы не спали: перепроверяли цепочку атаки, собирали технические доказательства и подробно описывали каждый этап, чтобы инженеры Telegram могли немедленно воспроизвести проблему. Уже 6 июня мы передали Telegram полный технический отчёт, явно обозначив его как конфиденциальный и запретив распространение без разрешения ExPatch LLC. Telegram контролировал приложение и мог быстрее всех перекрыть опасный путь — поэтому безопасность пользователей мы поставили выше сна, удобства и возможности сначала закрепить свой приоритет у Qt.

Telegram получил возможность защитить пользователей первым — и воспользовался ею. Однако вскоре соответствующее исправление появилось в Qt, тогда как нас не уведомили о координации, не подключили к взаимодействию с upstream, а имена исследователей и ExPatch так и не появились в публичной атрибуции. Была ли это обычная upstream-координация или нарушение доверия, благодаря которому Telegram вообще получил эту фору? В этой статье мы представим документы, код и хронологию, позволяющие читателю ответить на этот вопрос самостоятельно.

Исследование проводилось на Telegram Desktop 6.8.4, использовавшем Qt 6.11.1.

# От прокси к NTLM

Исследование началось с анализа архитектуры прокси-подсистемы. Мы установили, что обработку соединений через HTTP- и SOCKS5-прокси Telegram Desktop делегирует библиотеке Qt (https://www.qt.io/). На этом этапе нас интересовала не конкретная библиотека или компания, а сама граница доверия: что произойдёт, если прокси-сервер окажется враждебным и получит возможность управлять ответами во время установления соединения? Именно этот сценарий стал отправной точкой дальнейшего аудита.

Почти месяц мы исследовали этот механизм с разных сторон: проверяли границы доверия, переходы между состояниями и возможные способы навязать клиенту неожиданное поведение. Большинство направлений постепенно упиралось в стену, и поверхность атаки казалась почти исчерпанной. Переломный момент наступил, когда мы обнаружили неожиданное поведение Qt: прямо во время обмена с прокси-сервером библиотека принимала требование переключить аутентификацию на NTLM. Современное прокси-соединение могло незаметно перейти на устаревший механизм, десятилетиями связанный с атаками класса NTLM Relay. Именно этот переход полностью изменил направление исследования.

На верхнем уровне цепочка начинается предельно просто. Telegram Desktop устанавливает соединение через настроенный HTTP-прокси, а Qt отправляет ему запрос CONNECT. Враждебный прокси отвечает кодом 407 Proxy Authentication Required и указывает поддерживаемую схему: Proxy-Authenticate: NTLM. Qt воспринимает этот ответ как штатное требование аутентификации, автоматически запускает NTLM и отправляет первое сообщение рукопожатия — NTLM Type 1 (Negotiate). Всё это происходит без явного запроса согласия пользователя.

Следующим стал ключевой вопрос: какие проверки выполняет Qt перед тем, как доверить NTLM системные учётные данные пользователя? Поскольку атаки класса NTLM Relay известны уже несколько десятилетий, мы ожидали увидеть механизм, связывающий аутентификацию с конкретным прокси-сервером и не позволяющий повторно использовать тот же обмен на другом сервисе. Однако анализ реализации не выявил такой защиты: Qt не подтверждал ожидаемый SPN, не обеспечивал привязку к защищённому каналу через EPA и не требовал проверки целостности обмена посредством MIC. В результате контролирующий прокси атакующий мог не взламывать NTLM и не извлекать пароль — достаточно было переслать сообщения аутентификации другому сервису в реальном времени.

Эта находка стала переломной: давно забытый механизм NTLM оказался не спрятан за устаревшей корпоративной конфигурацией — его можно было активировать прямо во время обычного обмена с прокси-сервером. После подключения контролируемого HTTP-прокси атакующему требовался всего один штатно выглядящий ответ: 407 Proxy Authentication Required с заголовком Proxy-Authenticate: NTLM. Никакой сложной цепочки для запуска механизма не требовалось — один заголовок, и Qt автоматически вовлекал системные учётные данные пользователя в NTLM-обмен.

# Как Qt запускает NTLM без подтверждения

Чтобы понять, почему одного ответа прокси достаточно для запуска NTLM, мы проследили весь путь выполнения — от обработки ответа 407 до формирования сообщения аутентификации. Цепочка состоит из трёх уровней.

## Уровень 1 — Qt принимает ответ 407

Telegram Desktop настраивает прокси через QTcpSocket::setProxy. Затем в qhttpsocketengine.cpp:562–599 Qt обрабатывает ответ прокси на запрос CONNECT. При получении статуса 407 Proxy Authentication Required управление передаётся встроенному механизму аутентификации:

**qhttpsocketengine.cpp** — обработка 407 · строки 562–599

```
} else if (statusCode == 407) {                          // :562
    if (d->authenticator.isNull())
        d->authenticator.detach();

    priv = QAuthenticatorPrivate::getPrivate(
        d->authenticator
    );

    const auto headers = d->reply->header();

    priv->parseHttpResponse(
        headers,
        true,                                             // isProxy=true
        d->proxy.hostName()
    );                                                    // :567
}
```

На этом уровне NTLM ещё не выбран. Важно другое: Qt передаёт все полученные от прокси заголовки в parseHttpResponse() с параметром isProxy=true. Следовательно, дальнейший выбор схемы аутентификации зависит от содержимого контролируемого атакующим заголовка Proxy-Authenticate. Следующий уровень показывает, как значение NTLM превращается из текста в HTTP-ответе в команду запустить системную аутентификацию.

## Уровень 2 — Qt выбирает схему аутентификации

Решение о переходе на NTLM принимается внутри QAuthenticatorPrivate::parseHttpResponse() в qauthenticator.cpp:466–558. Поскольку функция была вызвана с параметром isProxy=true, Qt анализирует именно контролируемый прокси-сервером заголовок Proxy-Authenticate:

**qauthenticator.cpp** выбор схемы · строки 466–558

```
const auto search = isProxy
    ? QHttpHeaders::WellKnownHeader::ProxyAuthenticate
    : QHttpHeaders::WellKnownHeader::WWWAuthenticate;

// isProxy=true  → Proxy-Authenticate
// isProxy=false → WWW-Authenticate

for (const auto &current : headers.values(search)) {
    if (method < Basic
        && str.startsWith("basic"_L1, Qt::CaseInsensitive)) {
        method = Basic;

    } else if (method < Ntlm
               && str.startsWith("ntlm"_L1,
                                 Qt::CaseInsensitive)) {
        method = Ntlm;                                  // :488-490
        headerVal = QByteArrayView(current).mid(5);

    } else if (method < DigestMd5
               && str.startsWith("digest"_L1,
                                 Qt::CaseInsensitive)) {
        // ...

    } else if (method < Negotiate
               && str.startsWith("negotiate"_L1,
                                 Qt::CaseInsensitive)) {
        // ...
    }
}

// ...

case Ntlm:
case Negotiate:
    // work is done in calculateResponse()
    break;
```

Дополнительная проблема заключается в том, что перед разбором очередного ответа значение method сбрасывается. Поэтому прокси может потребовать NTLM не только в первом ответе, но и при последующем 407, изменив схему уже в процессе установления соединения. После выбора Ntlm функция не выполняет дополнительных проверок доверия к прокси — дальнейшее формирование системного ответа откладывается до вызова calculateResponse().

## Уровень 3 — Qt начинает NTLM-обмен без уведомления приложения

После выбора схемы управление возвращается в qhttpsocketengine.cpp:574–599:

qhttpsocketengine.cpp — запуск фазы NTLM · строки 574–599

```
1  if (priv->phase == QAuthenticatorPrivate::Done
2      || (priv->phase == QAuthenticatorPrivate::Start
3          && (priv->method == QAuthenticatorPrivate::Ntlm
4              || priv->method == QAuthenticatorPrivate::Negotiate))) {
5
6      if (priv->phase == QAuthenticatorPrivate::Start)
7          priv->phase = QAuthenticatorPrivate::Phase1;     // :578-580
8
9      if ((priv->method != QAuthenticatorPrivate::Ntlm
10         && priv->method != QAuthenticatorPrivate::Negotiate)
11         || credentialsWasSent) {
12
13         proxyAuthenticationRequired(
14             d->proxy,
15             &d->authenticator
16         );                                               // :596-597
17     }
18 }
```

Ключевым оказалось условие перед вызовом proxyAuthenticationRequired(). При обычной схеме Qt отправляет этот сигнал, позволяя приложению запросить учётные данные или отказаться от аутентификации. Но при первом переходе на NTLM или Negotiate, пока credentialsWasSent == false, всё условие становится ложным — сигнал не отправляется.

В результате Telegram Desktop не получает возможности подтвердить переход, предупредить пользователя или заблокировать его. Qt самостоятельно переводит аутентификатор в Phase1, вызывает calculateResponse(), обращается к Windows SSPI за системными SSO-учётными данными и отправляет прокси сообщение NTLM Type 1 (Negotiate). Именно поэтому переход происходит молча — не только для пользователя, но и для самого приложения.

Почему этот переход позволяет провести NTLM Relay? Критическая проблема заключается не в уязвимости самого SSPI, а в том, как Qt использует его внутри автоматической прокси-аутентификации. Если имя пользователя в QAuthenticator не задано, Qt не останавливает соединение и не запрашивает учётные данные у приложения. Вместо этого библиотека обращается к Windows SSPI и получает стандартные учётные данные текущей logon-сессии. В доменной среде это означает, что контролируемый атакующим прокси может без ведома пользователя инициировать отправку NTLM-токенов его корпоративной учётной записи.

## Этап 1 — calculateResponse() выбирает системную аутентификацию

Первое подтверждение находится в qauthenticator.cpp:585–592. Когда прокси требует NTLM, а challenge ещё не получен, Qt начинает первую фазу рукопожатия:

qauthenticator.cpp — calculateResponse() · строки 585–592

```
case QAuthenticatorPrivate::Ntlm:
    if (challenge.isEmpty()) {
#if QT_CONFIG(sspi)
        QByteArray phase1Token;

        if (user.isEmpty()) {
            // Only pull from system if no user was
            // specified in authenticator
            phase1Token = qSspiStartup(
                this,
                method,
                host
            );
        }

        // ...
#endif
    }

    // ...
    break;
```

Комментарий разработчиков Qt прямо описывает это поведение: Only pull from system if no user was specified in authenticator — если пользователь не был явно указан в аутентификаторе, учётные данные необходимо получить из системы. Именно здесь контролируемое прокси-сервером требование NTLM соединяется с учётной записью текущего пользователя Windows.

# Этап 2 — Системные учётные данные в qSspiStartup()

qauthenticator.cpp — qSspiStartup() · строки 1566–1607

```
SEC_WINNT_AUTH_IDENTITY auth;
auth.Flags = SEC_WINNT_AUTH_IDENTITY_UNICODE;

bool useAuth = false;

if (method == QAuthenticatorPrivate::Negotiate
    && !ctx->user.isEmpty()) {
    // Явные учётные данные используются только
    // для Negotiate с непустым user
    useAuth = true;
}

SECURITY_STATUS secStatus =
    pSecurityFunctionTable->AcquireCredentialsHandle(
        nullptr,
        L"NTLM",
        SECPKG_CRED_OUTBOUND,
        nullptr,
        useAuth ? &auth : nullptr,
        nullptr,
        nullptr,
        &ctx->sspiWindowsHandles->credHandle,
        &expiry
    );
```

Переменная useAuth становится истинной только для метода Negotiate, когда имя пользователя было задано явно. В пути NTLM функция передаёт в AcquireCredentialsHandle() значение nullptr вместо структуры SEC*WINNT*AUTH_IDENTITY. Согласно модели SSPI, это означает использование стандартных исходящих учётных данных текущего контекста безопасности Windows. В доменной среде ими обычно становятся учётные данные вошедшего в систему корпоративного пользователя.

При этом Qt не получает пароль или NTLM-хеш в открытом виде. SSPI возвращает непрозрачный credential handle, с помощью которого далее формируются сообщения NTLM-аутентификации. Это важное уточнение: уязвимость позволяет атакующему перенаправить готовую аутентификацию пользователя.

## Этап 3 — NTLM-контекст не привязывается к ожидаемому прокси

Полученный credential handle передаётся в qSspiContinue(), где Qt вызывает Windows SSPI для продолжения рукопожатия:

qauthenticator.cpp qSspiContinue()

```
1  InitializeSecurityContext(
2      &ctx->sspiWindowsHandles->credHandle,
3      ...,
4      ISC_REQ_ALLOCATE_MEMORY, // единственный запрошенный флаг
5      ...
6  );
```

В этом вызове Qt запрашивает только автоматическое выделение памяти для выходного токена. При этом отсутствуют три механизма, способные затруднить перенаправление аутентификации:

- флаг ISC*REQ*INTEGRITY не установлен, поэтому Qt не требует защиты целостности создаваемого контекста;
- буфер SECBUFFER*CHANNEL*BINDINGS не передаётся, а значит NTLM-обмен не связывается с конкретным защищённым каналом через EPA/Channel Binding;
- для прямого метода NTLM значение targetNameW остаётся пустым, поэтому контекст не привязывается к ожидаемому сервису посредством SPN.

В совокупности это означает, что сформированный SSPI-контекст не связан с тем прокси-сервером, которому, как полагает пользователь, он аутентифицируется. В протестированной конфигурации Qt генерировал сообщения NTLM Type 1 и NTLM Type 3 с ответом доменного пользователя, которые контролирующий прокси мог в реальном времени перенаправить другому корпоративному сервису.

# Воздействие и поверхность атаки

Масштаб воздействия оказался значительным: потенциальной точкой входа становилась корпоративная Windows-система, на которой пользователь вошёл под доменной учётной записью. Это не экзотическая лабораторная конфигурация, а распространённый класс корпоративных устройств.

Практический результат — аутентифицированная сессия на уязвимой relay-цели в пределах прав доменного пользователя. В зависимости от выбранного сервиса и привилегий жертвы последствия могли начинаться с доступа к корпоративным данным и заканчиваться удалённым выполнением кода или компрометацией доменной инфраструктуры. И всё это запускалось одним ответом 407 Proxy Authentication Required.

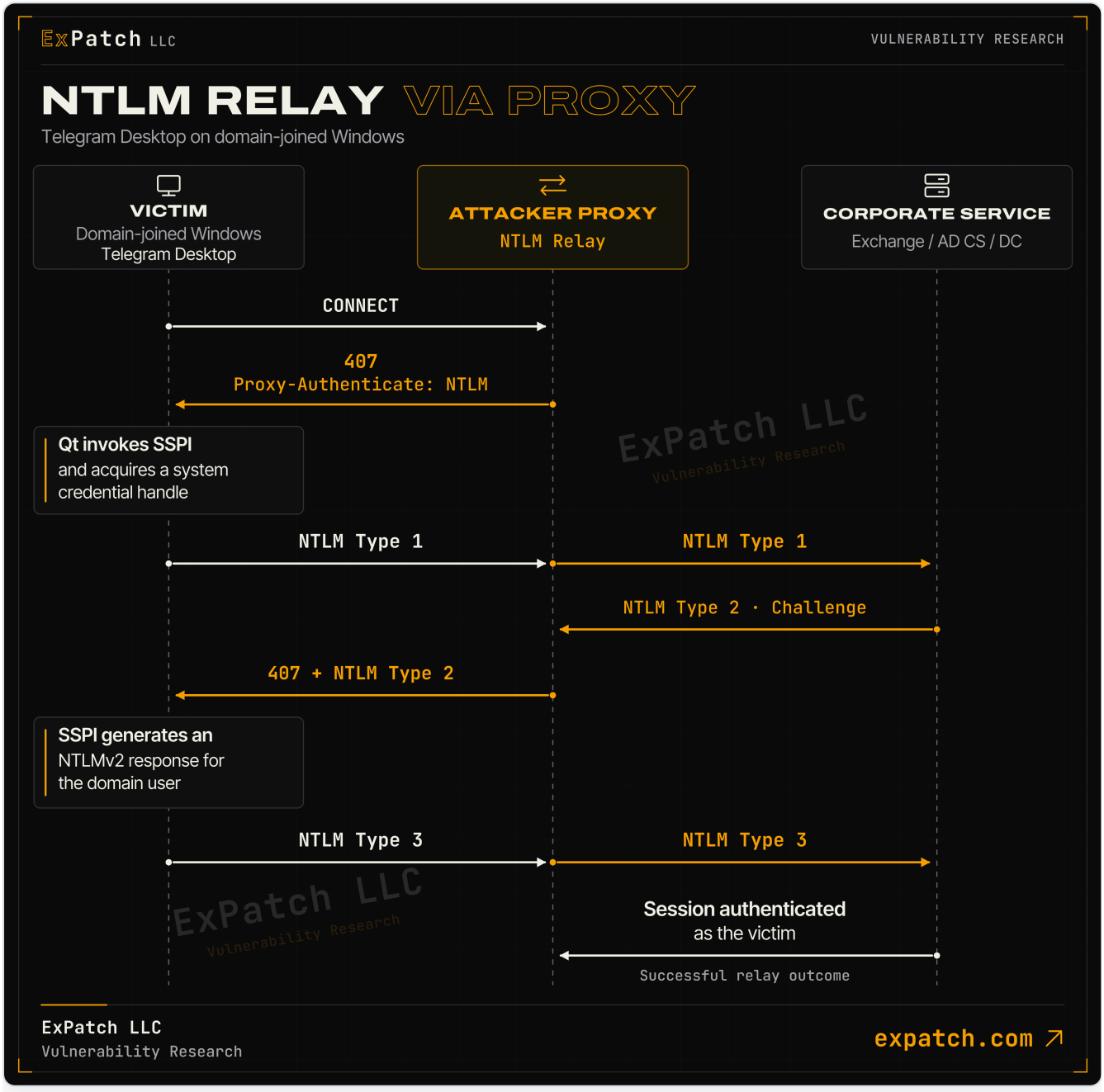



**Рис. 2.** *Схема NTLM Relay: прокси передаёт NTLM-сообщения между Telegram Desktop и корпоративным сервисом, получая аутентифицированную сессию с правами жертвы.*

## Двухэтапная эскалация NTLM → Negotiate (P1): два NTLMv2-ответа за один сеанс

Даже явное указание логина и пароля для прокси не защищало пользователя. Контролируемый сервер мог провести две последовательные аутентификации в рамках одного соединения:

**Схема обмена** NTLM → Negotiate

```
1. 407 + Proxy-Authenticate: NTLM
   |
   └─► Qt использует явно заданные учётные данные прокси
       и формирует NTLMv2-ответ №1

2. Прокси отклоняет первую аутентификацию
   |
   └─► qhttpsocketengine сбрасывает authenticator
       user становится пустым

3. 407 + Proxy-Authenticate: Negotiate
   |
   └─► qSspiStartup() запрашивает стандартные credentials SSPI
       Negotiate переключается на NTLM
       Qt формирует NTLMv2-ответ №2
       от имени доменного пользователя Windows
```

Таким образом, атакующий получал за один сеанс два различных NTLMv2-ответа: первый — на основе явно заданных учётных данных прокси, второй — на основе системной учётной записи Windows. Оба ответа оказывались под контролем атакующего, а ответ доменного пользователя мог быть немедленно перенаправлен выбранному корпоративному сервису в рамках NTLM Relay.

Этот сценарий существенно расширяет класс затронутых пользователей. Уязвимым оказывался не только клиент с пустым полем username: отклонив первую аутентификацию и навязав повторный обмен через Negotiate, прокси мог открыть путь к системным credentials даже после того, как пользователь явно указал отдельный логин и пароль для прокси.

## Вектор SOCKS5-in-tunnel (V3): внедрение 407 внутрь активного туннеля

Вектор V3 не требовал перехода по tg://proxy-ссылке. Он затрагивал доменные Windows-системы, на которых корпоративный прокси централизованно настраивался через GPO, WPAD или PAC-файл.

Ключевое отличие V3 от базового вектора заключалось в точке внедрения ответа 407. После установления SOCKS5-туннеля контролируемый прокси видел незашифрованный HTTP-запрос Telegram Desktop — включая заголовки, URL и тело MTProto POST — и мог подменить ответ сервера:

**HTTP** ответ внутри SOCKS5-туннеля

```
HTTP/1.1 407 Proxy Authentication Required
Proxy-Authenticate: NTLM
```

Qt интерпретировал этот ответ как штатное требование прокси-аутентификации и автоматически запускал SSPI-путь.

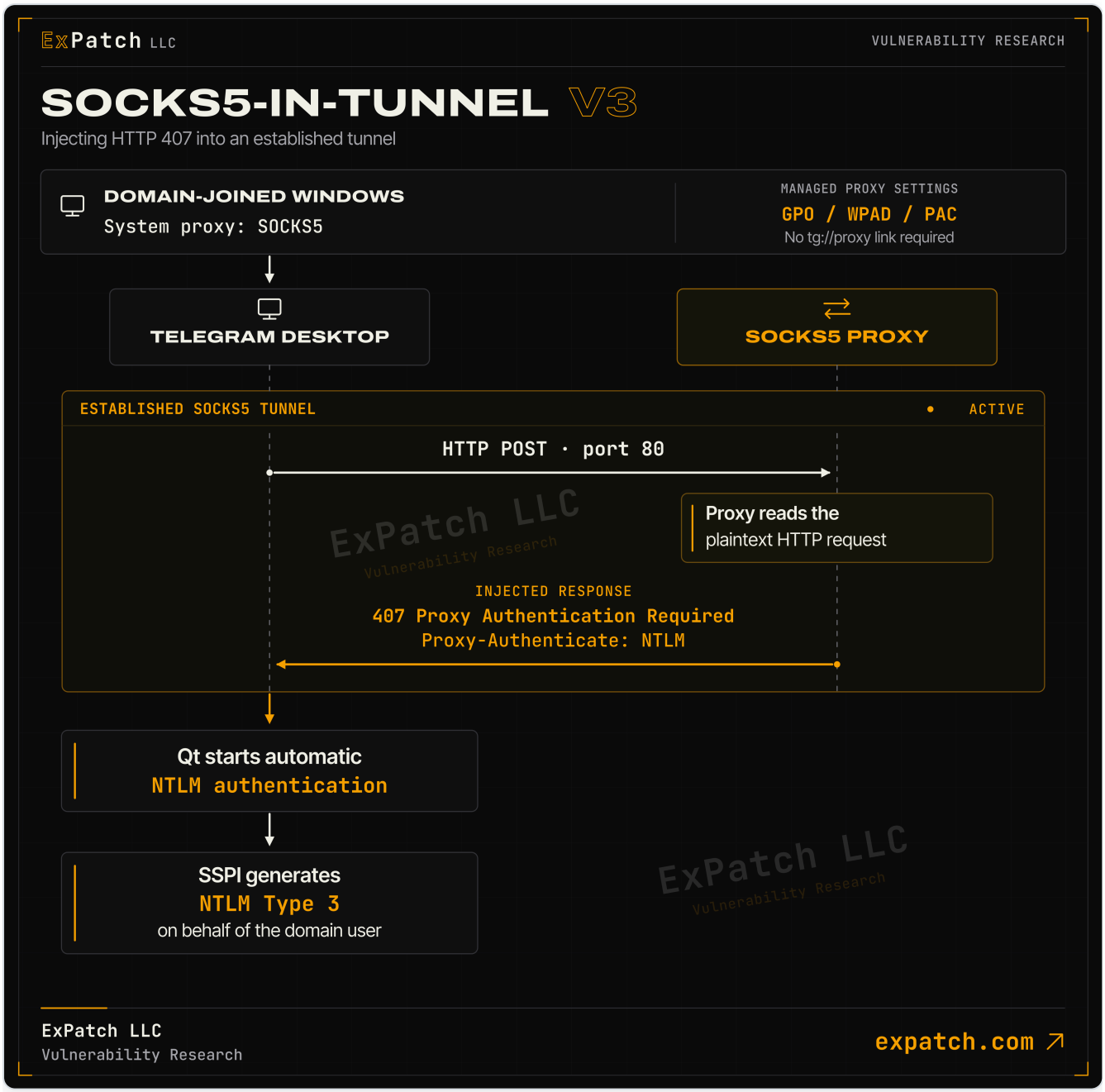



**Рис. 3.** Вектор V3: контролируемый SOCKS5-прокси внедряет 407 Proxy Authentication Required внутрь активного туннеля, после чего Qt автоматически получает через SSPI NTLMv2-ответ доменного пользователя.

В лабораторной среде с Windows 10, присоединённой к домену, одна тестовая сессия Telegram Desktop сформировала более 230 сообщений NTLM Type 3. Повторные попытки Qt многократно увеличивали количество доступного атакующему аутентификационного материала и возможностей для его перенаправления.

## Корень уязвимости находится в QtNetwork

Дальнейший анализ показал, что Telegram Desktop был лишь одним из способов достижения уязвимого кода. Первопричина находилась в общей реализации HTTP-аутентификации модуля QtNetwork — qauthenticator.cpp — и не ограничивалась ответами 407 Proxy Authentication Required.

Тот же путь можно было запустить без прокси. Если контролируемый атакующим HTTP-сервер отвечал на обычный незашифрованный запрос кодом 401 Unauthorized и заголовком:

HTTP прямой серверный вектор

```
WWW-Authenticate: NTLM
```

Qt воспринимал его как штатное требование аутентификации. На присоединённой к домену Windows-системе, при отсутствии явно заданных credentials, библиотека без диалога с пользователем обращалась к SSPI и начинала NTLM-обмен от имени текущей учётной записи.

Таким образом, прокси был не обязательным условием эксплуатации, а лишь особенно опасной точкой входа в Telegram Desktop. Уязвимый механизм находился глубже — в общем HTTP-клиентском стеке Qt, которым пользуется множество приложений.

## Где ещё достигается уязвимый код

Мы не ограничились анализом исходного кода Qt и Telegram Desktop. Это была уже не теоретическая экстраполяция по общему API: тот же уязвимый примитив мы практически воспроизвели в qBittorrent, Nextcloud Desktop и Electrum.

### Практически подтверждённые приложения

- qBittorrent 4.6.3 с Qt 6.4.2. Контролируемый прокси переключил приложение с Basic на NTLM ответом 407 Proxy Authentication Required — без уведомления пользователя. Полный NTLM-handshake завершился формированием NtChallengeResponse размером 65 076 байт, из которых около 64 876 байт приходилось на посторонние данные из heap.

- Nextcloud Desktop 3.11.0 с Qt 5.15.13. За одну тестовую сессию приложение выполнило четыре последовательных NTLM-обмена. Каждый ответ достигал 65 076 байт, поэтому суммарно за одну сессию раскрывалось около 260 КБ памяти. В дампах находились полные HTTP-запросы, WebDAV- и API-маршруты Nextcloud, служебные заголовки и NTLM-токены предыдущих обменов.

- Electrum 4.7.2 с Qt 6.11.0. Здесь обнаружился отдельный прямой путь, не требовавший прокси. Контролируемое описание Bitcoin- или LNURL-счёта отображалось QML-компонентом как rich text, а внедрённый `<img src="http://…">` заставлял QNetworkAccessManager выполнить запрос к серверу атакующего. В ответ на 401 WWW-Authenticate: NTLM Qt автоматически отправлял неподписанный NTLM Type 1, причём Electrum не подключал обработчик authenticationRequired, способный остановить этот обмен. Достижимость уязвимого кода и

автоматическое начало NTLM-аутентификации были подтверждены экспериментально; завершение Type 3 через SSPI и последующий relay требуют Windows-машины в доменном окружении.

Таким образом, речь шла не об изолированной ошибке Telegram Desktop и не о предположении, основанном только на чтении исходного кода. Один дефект QtNetwork был практически воспроизведён в нескольких независимых продуктах, разных версиях Qt и двух точках входа — через ответы 401 и 407. Telegram оставался главным объектом нашего исследования, однако фактический радиус воздействия обнаруженной первопричины оказался значительно шире одного приложения.

# Второй примитив: раскрытие heap-памяти

Масштаб проблемы оказался значительно шире Telegram, однако именно Telegram оставался главной целью нашего исследования. Мы решили не останавливаться на принудительной NTLM-аутентификации и relay-атаке. Если недоверенный сервер мог без участия пользователя активировать редко используемый код аутентификации и управлять содержимым NTLM Type 2, следовало проверить, насколько безопасно Qt разбирает остальные контролируемые атакующим поля.

Именно там мы обнаружили второй независимый примитив — чтение неинициализированной heap-памяти в qExtractServerTime():

qauthenticator.cpp — qExtractServerTime()

```
1  static QByteArray qExtractServerTime(const QByteArray &targetInfoBuff)
2  {
3    QByteArray timeArray;
4    QDataStream ds(targetInfoBuff);
5    ds.setByteOrder(QDataStream::LittleEndian);
6    quint16 avId, avLen;
7
8    ds >> avId;
9    ds >> avLen;
10   while (avId != 0) {
11       if (avId == AVTIMESTAMP) {
12           timeArray.resize(avLen);                // длина контролируется сервером
13           ds.readRawData(timeArray.data(), avLen); // результат короткого чтения не проверяется
14       }
15       // ...
16   }
17 }
```

В NTLM Type 2 сервер управляет содержимым targetInfo, включая идентификатор MsvAvTimestamp и 16-битное поле avLen. Мы указывали длину 65 000 байт, но передавали только восьмибайтовое значение timestamp. Qt выделял буфер заявленного размера, записывал в него восемь доступных байт и не проверял, что readRawData() выполнил короткое чтение. Оставшиеся 64 992 байта сохраняли прежнее содержимое heap. Затем Qt включал весь буфер — вместе с неинициализированной областью — в NTLMv2-ответ и отправлял его серверу атакующего.

Это превратило обычный NTLM-обмен в удалённый и повторяемый примитив чтения памяти. В полученных дампах мы подтвердили утечку указателей кода и метаданных heap, достаточных для обхода ASLR, сетевых адресов, названий VPN-клиентов и адаптеров, HTTP-заголовков, а также предыдущих NTLM-токенов. За 100 повторных обменов, занимавших около пяти минут, прокси мог получить приблизительно 6,5 МБ памяти процесса Telegram Desktop.


**Рис. 4.** Механизм утечки: Qt выделяет 65 000 байт по контролируемому avLen, записывает только восемь байт timestamp и включает оставшиеся 64 992 байта heap-памяти в NTLM Type 3.

## Что находилось в утекшей памяти

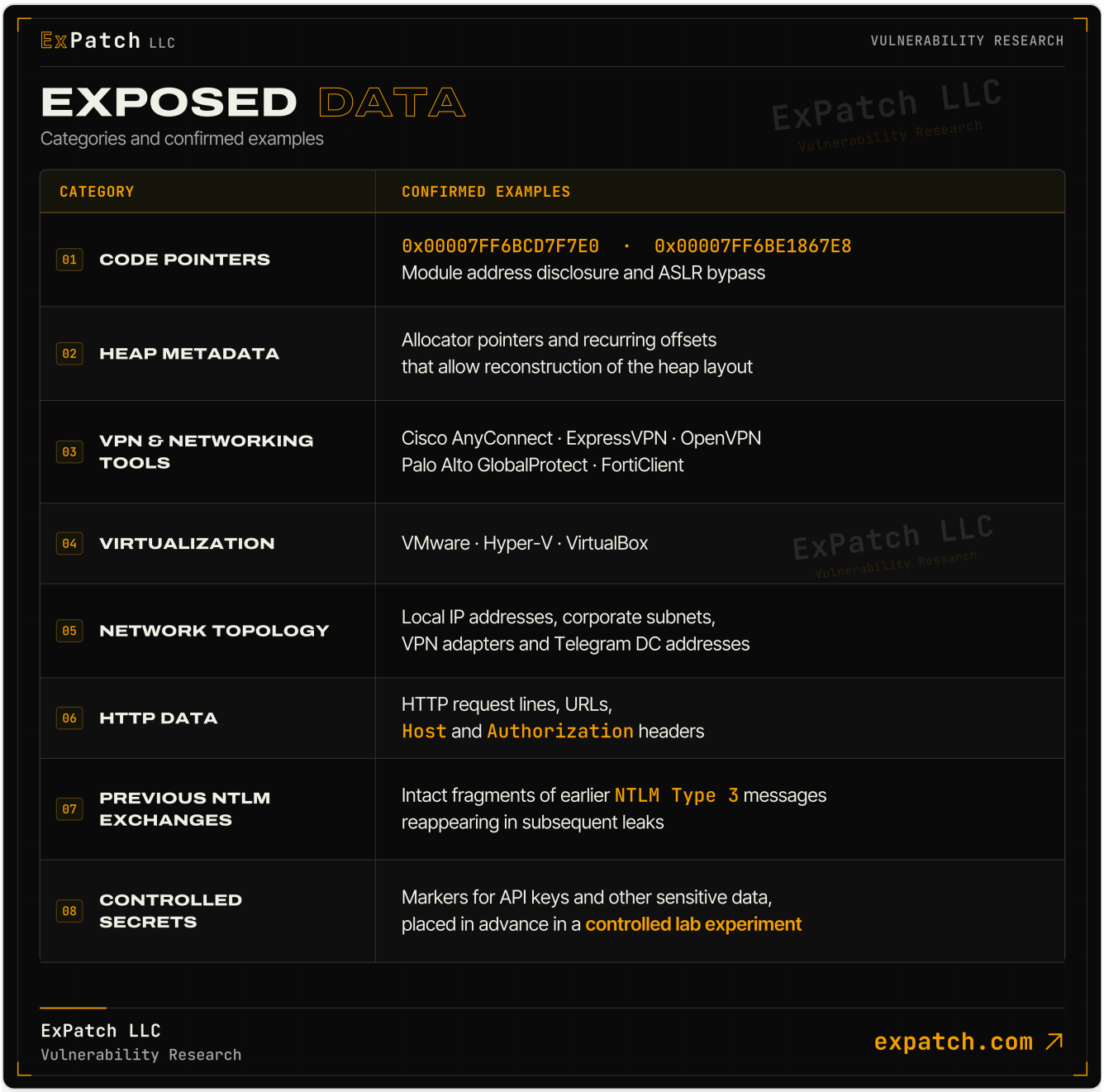



**Рис. 5.** Категории данных, обнаруженные в утекшей памяти в ходе лабораторных испытаний.

По отдельности эти фрагменты могли выглядеть случайным мусором. Вместе они образовывали подробный профиль устройства и его владельца: имя пользователя, рабочий домен и hostname из NTLM, используемый VPN-клиент, виртуализацию, внутреннюю адресацию и корпоративную сетевую инфраструктуру.

Но последствия не ограничивались деанонимизацией. Уязвимость без разбора передавала содержимое повторно используемых heap-буферов. В ходе контролируемого heap-grooming мы заранее помещали в них тестовые пароли и маркеры API-секретов, а затем обнаруживали эти значения в NTLMv2-ответе, отправленном прокси. По тому же механизму могли раскрываться API-ключи, токены сессий, cookie, заголовки Authorization и другие секреты, недавно находившиеся в памяти процесса. В последующих утечках также появлялись ранее сформированные NTLM Type 3.

Важно: пароль доменного пользователя не извлекался непосредственно из NTLM. Открытые пароли могли утекать как содержимое повторно использованного буфера, независимо от самого NTLM-протокола.

Особенно тревожно, что каналом утечки становилась функция, предназначенная помогать пользователю обходить ограничения и сохранять доступ к информации. Прокси продолжал работать, Telegram не показывал предупреждений, а пользователь не видел, что оператору вредоносного прокси последовательно передаются фрагменты памяти — включая пароли, API-секреты и данные, способные раскрыть его личность или предоставить доступ к связанным системам.

# Полная RCE-цепочка: лабораторное подтверждение

Чтобы проверить максимальный практический ущерб, мы построили изолированный стенд с доменом Active Directory, Windows Server 2022 и Enterprise CA с доступным AD CS Web Enrollment.

На стороне жертвы использовался официальный Telegram Desktop 6.8.2 без каких-либо изменений исходного кода, исполняемого файла, Qt или механизмов аутентификации. Клиент работал в штатной конфигурации. Единственным действием пользователя было добавление прокси предусмотренным Telegram способом. Вся атакующая логика находилась на стороне контролируемого прокси и лабораторной инфраструктуры.

После подключения Telegram к прокси цепочка выполнялась автоматически:


**Рис. 6.** Полная лабораторная цепочка: от добавления tg://proxy до получения SYSTEM на узлах, где скомпрометированная учётная запись обладала административными правами.

На стенде все 9 из 9 попыток relay к AD CS завершились успешной аутентификацией и выпуском сертификата. Поскольку тестовая учётная запись обладала правами Domain Admin, мы получили NT AUTHORITY\SYSTEM как на рабочей станции, так и на контроллере домена. Полная цепочка занимала около 30–35 секунд и не требовала подбора или восстановления пароля.

Уровень последующего доступа определялся правами скомпрометированной учётной записи: обычный пользователь не превращался автоматически в администратора домена. Однако relay учётной записи администратора, привилегированного сотрудника или сервисного аккаунта мог непосредственно привести к удалённому выполнению кода и компрометации соответствующих систем. Наш эксперимент подтвердил верхнюю границу воздействия: полный захват домена через штатный, немодифицированный Telegram Desktop после добавления контролируемого прокси.

Выданный сертификат создавал отдельный механизм сохранения доступа. Смена пароля не отзывала его автоматически: при сроке действия сертификата в один–два года атакующий мог продолжать проходить аутентификацию до истечения срока, явного отзыва сертификата или блокировки учётной записи.

## Почему это уязвимость Telegram Desktop, а не только Qt

Сводить вопрос к тому, в каком исходном файле находилась ошибка, — значит игнорировать всю цепочку эксплуатации. Библиотечная первопричина действительно находилась в QtNetwork. Но атака начиналась не в абстрактном приложении Qt: она начиналась в официальном Telegram Desktop, после добавления прокси штатным механизмом tg://proxy.

Именно Telegram Desktop делал дефект практически достижимым:

- принимал и сохранял предоставленный пользователем прокси;
- передавал его в Qt без запрета NTLM и Negotiate;
- не предоставлял пользователю возможности увидеть или отклонить выбранную прокси-сервером схему;
- позволял недоверенному прокси активировать SSPI и получить NTLMv2-ответ;
- передавал через этот же канал фрагменты памяти процесса Telegram Desktop.

Это не просто присутствие уязвимой зависимости. Уберите любой из ключевых продуктовых элементов — tg://proxy, автоматическое сохранение прокси или отсутствие запрета NTLM — и описанная цепочка через Telegram прекращается. Следовательно, решения Telegram были необходимой частью эксплуатации, а не случайным окружением библиотечной ошибки.

# Ответственное раскрытие и хронология

## 6 июня: полный отчёт передан Telegram

После ночи непрерывной проверки и подготовки материалов 6 июня 2026 года мы направили Telegram не краткое описание и не предварительную гипотезу, а законченный 21-страничный технический отчёт EXPATCH-2026-TG-002. В переданный пакет входили анализ исходного кода, условия эксплуатации, PoC, видеодемонстрация, результаты лабораторных испытаний и полная RCE-цепочка.

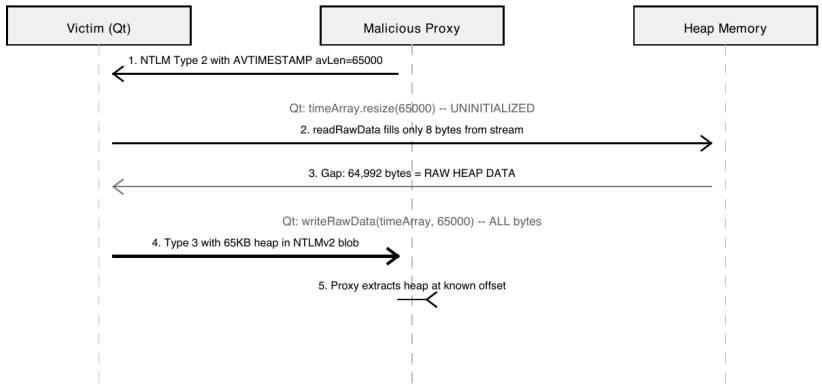
Отдельный раздел отчёта уже тогда описывал утечку памяти в qExtractServerTime(): контролируемое сервером поле MsvAvTimestamp.avLen, увеличение QByteArray через timeArray.resize(avLen), неполное чтение readRawData() и последующую отправку непрочитанного содержимого heap внутри NTLMv2-ответа. На странице 20 мы предложили конкретное исправление: проверять avLen по оставшемуся размеру буфера и принимать timestamp только ожидаемой длины — восемь байт.

## 65KB Heap Information Disclosure (F-QT-A14)

Each NTLM exchange leaks up to 65,535 bytes of uninitialized heap memory from the victim's Telegram Desktop process. The leaked data is sent DIRECTLY (not hashed) in the NTLMv2 response blob to the proxy. The proxy decodes base64 and reads raw heap bytes at a known offset.

### How the Heap Leak Works

65KB Heap Leak Mechanism




### Vulnerable Code Path (qauthenticator.cpp)

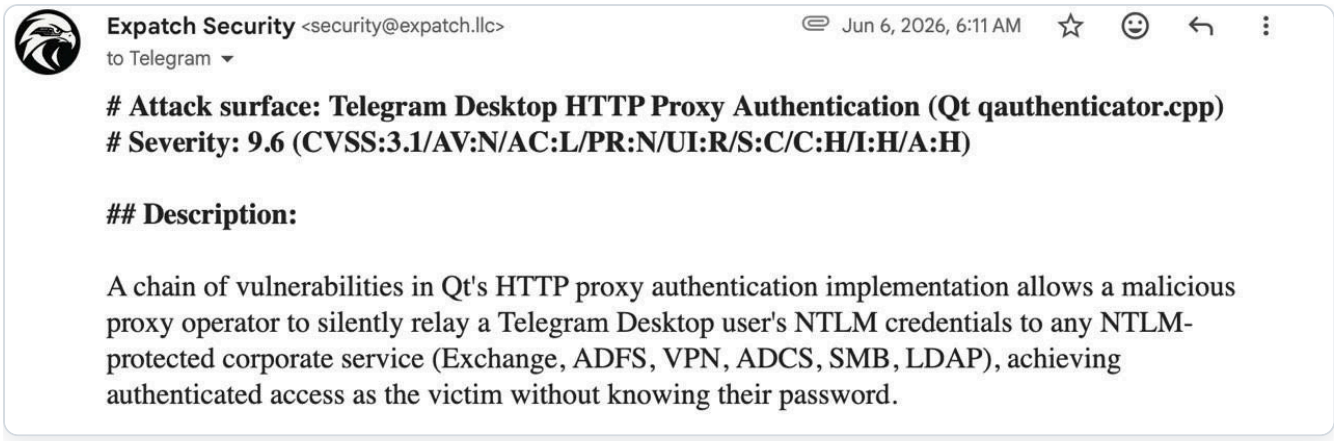
```
// Line 1330: Qt parses NTLM Type 2 targetInfo AV_PAIRs
if (avId == MsvAvTimestamp) {         // avId=7 from proxy

    // Line 1343: ALLOCATE -- avLen from proxy (uint16, up to 65535)
    QByteArray timeArray;
    timeArray.resize(avLen);           // <<< UNINITIALIZED HEAP!

    // Line 1345: READ -- fills only N bytes remaining in stream
    ds.readRawData(timeArray.data(), avLen);
    // Stream has ~8 bytes left. Gap = avLen - 8 = 64,992 bytes
    // Those 64,992 bytes contain WHATEVER WAS IN HEAP BEFORE

    // Line 1384: WRITE -- sends ALL bytes including uninitialized gap
    ds.writeRawData(timeArray.constData(), timeArray.size());
    // Entire timeArray goes into NTLMv2 response temp blob

    // Line 1417: SEND -- blob sent as base64 in Proxy-Authorization header
    ntChallengeResp.append(temp);
    // >>> 65KB of RAW HEAP sent to proxy in cleartext <<<
}
```

### Proven Leaked Data (Live Telegram Desktop 6.8.2)

| Data Category | Examples Found in Heap Dumps |
|---|---|
| Network Adapters | Intel(R) Wi-Fi 6E AX211 160MHz, VMware VMnet1/VMnet8 |
| Virtualization | Hyper-V Virtual Ethernet Adapter, VirtualBox Host-Only |
| VPN Client Name | ExpressVPN Packet Filter Driver, OpenVPN Data Channel Offload |
| VPN Adapter Config | Cisco AnyConnect, Palo Alto GlobalProtect, FortiClient SSL VPN |
| Local IPs (private) | 192.168.0.1, 192.168.0.6, 10.0.0.1, 172.19.0.1, 127.0.0.1 |
| Public/External IPs | Telegram DC IPs: 149.154.167.41, 91.108.56.172 |



**Рис. 7.** Страница 5 отчёта EXPATCH-2026-TG-002. В переданном 6 июня документе уже были названы уязвимая функция, управляемое поле avLen, механизм раскрытия 64 992 байт heap-памяти и результаты проверки на штатном Telegram Desktop 6.8.2.



**Expatch Security** <security@expatch.llc> Jun 6, 2026, 6:11 AM

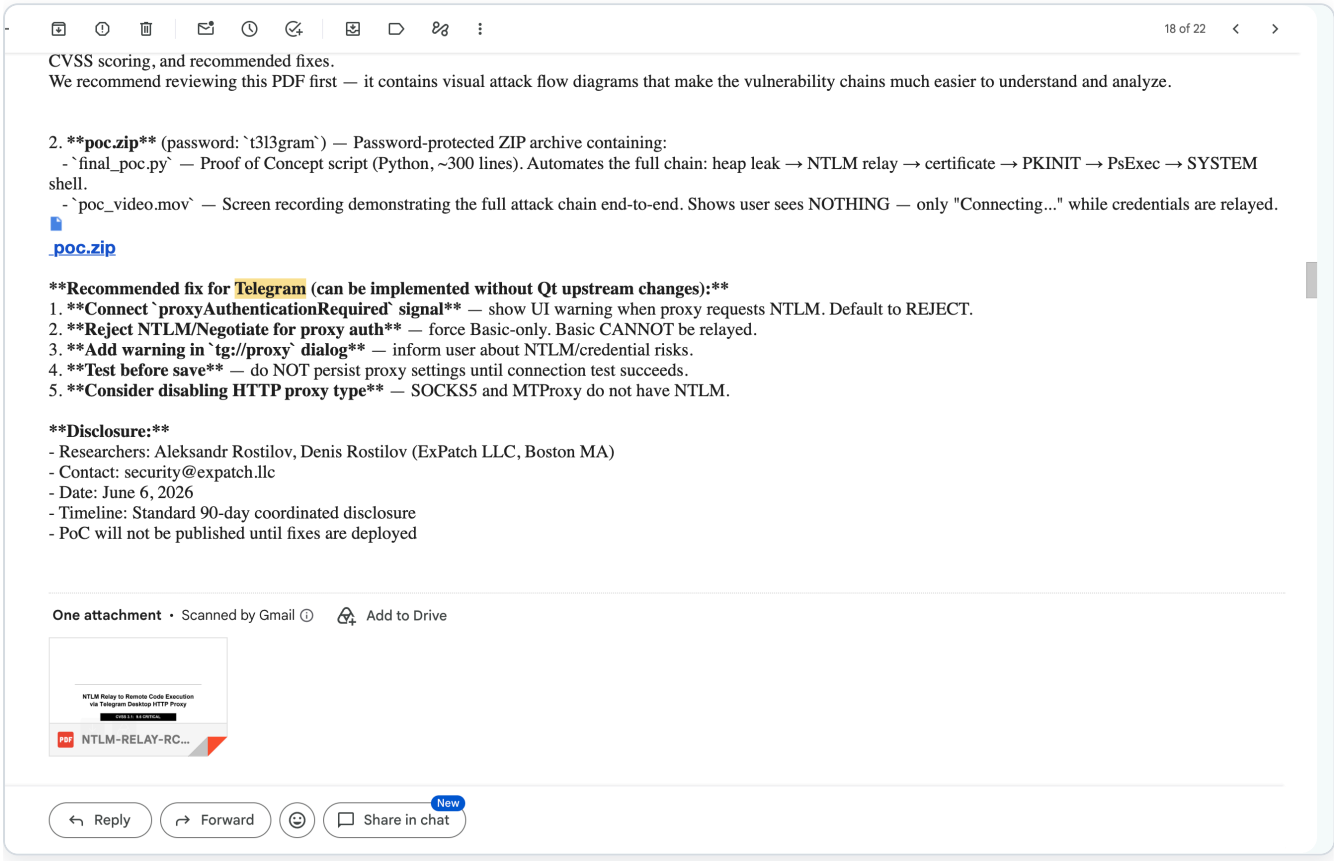
to Telegram

**# Attack surface: Telegram Desktop HTTP Proxy Authentication (Qt qauthenticator.cpp)**
**# Severity: 9.6 (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H)**

**## Description:**

A chain of vulnerabilities in Qt's HTTP proxy authentication implementation allows a malicious proxy operator to silently relay a Telegram Desktop user's NTLM credentials to any NTLM-protected corporate service (Exchange, ADFS, VPN, ADCS, SMB, LDAP), achieving authenticated access as the victim without knowing their password.

**Рис. 8.** Начало исходного обращения ExPatch в Telegram от 6 июня 2026 года.



18 of 22

CVSS scoring, and recommended fixes.
We recommend reviewing this PDF first — it contains visual attack flow diagrams that make the vulnerability chains much easier to understand and analyze.

2. **poc.zip** (password: `t3l3gram`) — Password-protected ZIP archive containing:
- `final_poc.py` — Proof of Concept script (Python, ~300 lines). Automates the full chain: heap leak → NTLM relay → certificate → PKINIT → PsExec → SYSTEM shell.
- `poc_video.mov` — Screen recording demonstrating the full attack chain end-to-end. Shows user sees NOTHING — only "Connecting..." while credentials are relayed.

poc.zip

**Recommended fix for Telegram (can be implemented without Qt upstream changes):**
1. **Connect `proxyAuthenticationRequired` signal** — show UI warning when proxy requests NTLM. Default to REJECT.
2. **Reject NTLM/Negotiate for proxy auth** — force Basic-only. Basic CANNOT be relayed.
3. **Add warning in `tg://proxy` dialog** — inform user about NTLM/credential risks.
4. **Test before save** — do NOT persist proxy settings until connection test succeeds.
5. **Consider disabling HTTP proxy type** — SOCKS5 and MTProxy do not have NTLM.

**Disclosure:**
- Researchers: Aleksandr Rostilov, Denis Rostilov (ExPatch LLC, Boston MA)
- Contact: security@expatch.llc
- Date: June 6, 2026
- Timeline: Standard 90-day coordinated disclosure
- PoC will not be published until fixes are deployed

One attachment • Scanned by Gmail Add to Drive

NTLM Relay to Remote Code Execution via Telegram Desktop HTTP Proxy

NTLM-RELAY-RC...

Reply Forward Share in chat New

**Рис. 9.** Продолжение письма от 6 июня. На скриншоте видны переданные материалы, рекомендации, которые Telegram мог реализовать без ожидания upstream-изменений, сведения об авторах и вложенный отчёт.

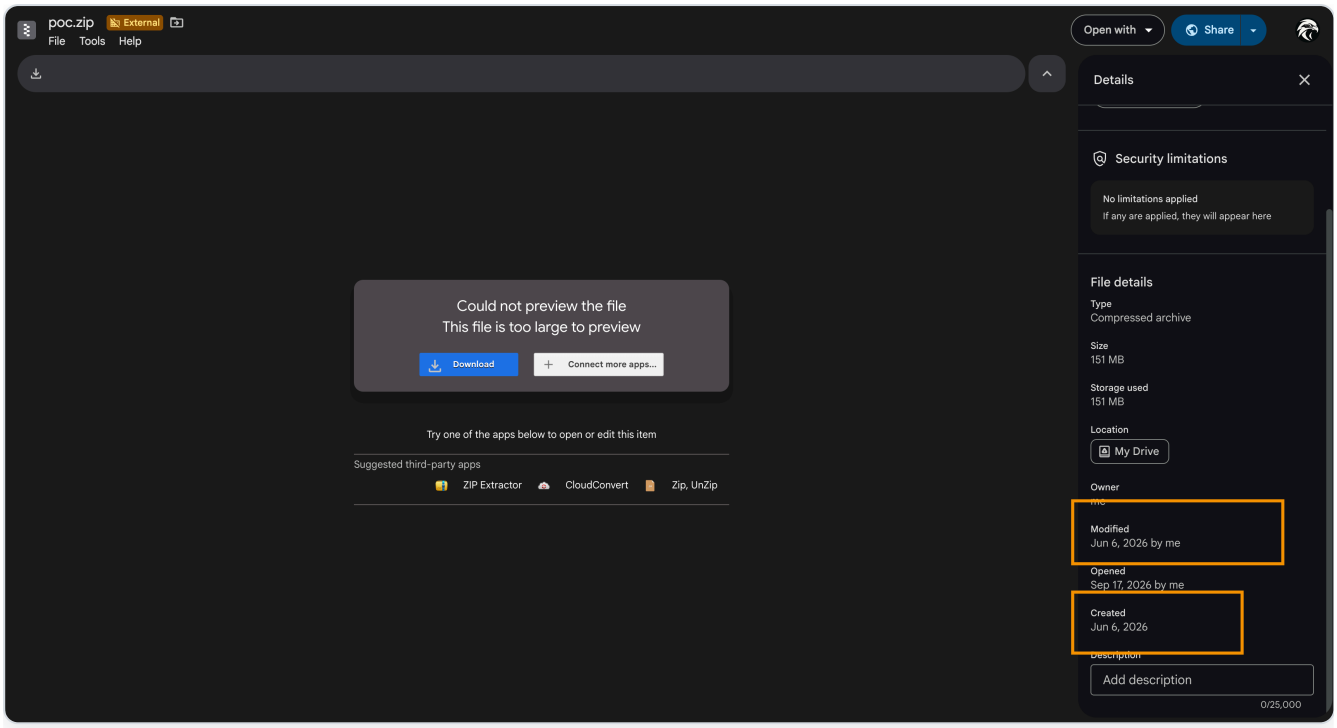


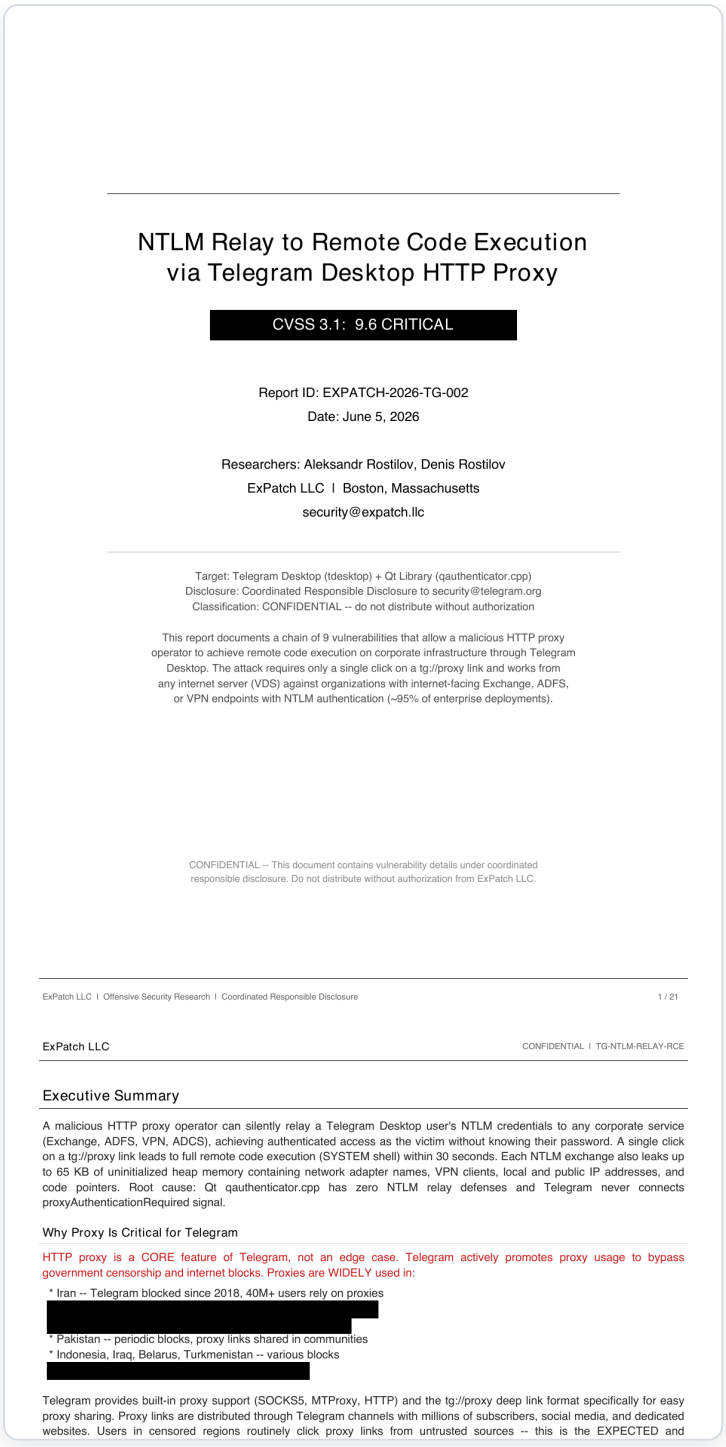

**Рис. 10.** Свойства переданного архива poc.zip: Google Drive датирует его создание и последнее изменение 6 июня 2026 года.

Само по себе это изображение не доказывает скачивание архива получателем. Однако вместе со скриншотом письма, содержащим ссылку на poc.zip, оно подтверждает, что полный PoC существовал и был доступен Telegram уже в день обращения.

Хронологию дополнительно подтверждают метаданные исходного PDF. Его обложка датирована 5 июня, а поле CreationDate содержит время 6 июня 2026 года, 04:04:01 UTC — примерно за шесть часов до отправки показанного письма. SHA-256 файла:

> **SHA-256**
>
> `c2c88a4db92a9fb06cd64a2efec3e337714a0ecdc32526c6a65b4dc477173f6c`

На обложке документ был прямо обозначен как CONFIDENTIAL и передан в рамках Coordinated Responsible Disclosure с требованием не распространять его без разрешения ExPatch LLC. Мы обратились к Telegram раньше, чем завершили отдельное обращение в Qt, поскольку Telegram мог немедленно перекрыть наиболее опасный путь на уровне собственного приложения.



NTLM Relay to Remote Code Execution
via Telegram Desktop HTTP Proxy

CVSS 3.1: 9.6 CRITICAL

Report ID: EXPATCH-2026-TG-002
Date: June 5, 2026

Researchers: Aleksandr Rostilov, Denis Rostilov
ExPatch LLC I Boston, Massachusetts
security@expatch.llc

Target: Telegram Desktop (tdesktop) + Qt Library (qauthenticator.cpp)
Disclosure: Coordinated Responsible Disclosure to security@telegram.org
Classification: CONFIDENTIAL -- do not distribute without authorization

This report documents a chain of 9 vulnerabilities that allow a malicious HTTP proxy operator to achieve remote code execution on corporate infrastructure through Telegram Desktop. The attack requires only a single click on a tg://proxy link and works from any internet server (VDS) against organizations with internet-facing Exchange, ADFS, or VPN endpoints with NTLM authentication (~95% of enterprise deployments).

CONFIDENTIAL -- This document contains vulnerability details under coordinated responsible disclosure. Do not distribute without authorization from ExPatch LLC.

ExPatch LLC I Offensive Security Research I Coordinated Responsible Disclosure 1 / 21

ExPatch LLC CONFIDENTIAL I TG-NTLM-RELAY-RCE

Executive Summary

A malicious HTTP proxy operator can silently relay a Telegram Desktop user's NTLM credentials to any corporate service (Exchange, ADFS, VPN, ADCS), achieving authenticated access as the victim without knowing their password. A single click on a tg://proxy link leads to full remote code execution (SYSTEM shell) within 30 seconds. Each NTLM exchange also leaks up to 65 KB of uninitialized heap memory containing network adapter names, VPN clients, local and public IP addresses, and code pointers. Root cause: Qt qauthenticator.cpp has zero NTLM relay defenses and Telegram never connects proxyAuthenticationRequired signal.

Why Proxy Is Critical for Telegram

HTTP proxy is a CORE feature of Telegram, not an edge case. Telegram actively promotes proxy usage to bypass government censorship and internet blocks. Proxies are WIDELY used in:

* Iran -- Telegram blocked since 2018, 40M+ users rely on proxies
[redacted]
* Pakistan -- periodic blocks, proxy links shared in communities
* Indonesia, Iraq, Belarus, Turkmenistan -- various blocks
[redacted]

Telegram provides built-in proxy support (SOCKS5, MTProxy, HTTP) and the tg://proxy deep link format specifically for easy proxy sharing. Proxy links are distributed through Telegram channels with millions of subscribers, social media, and dedicated websites. Users in censored regions routinely click proxy links from untrusted sources -- this is the EXPECTED and

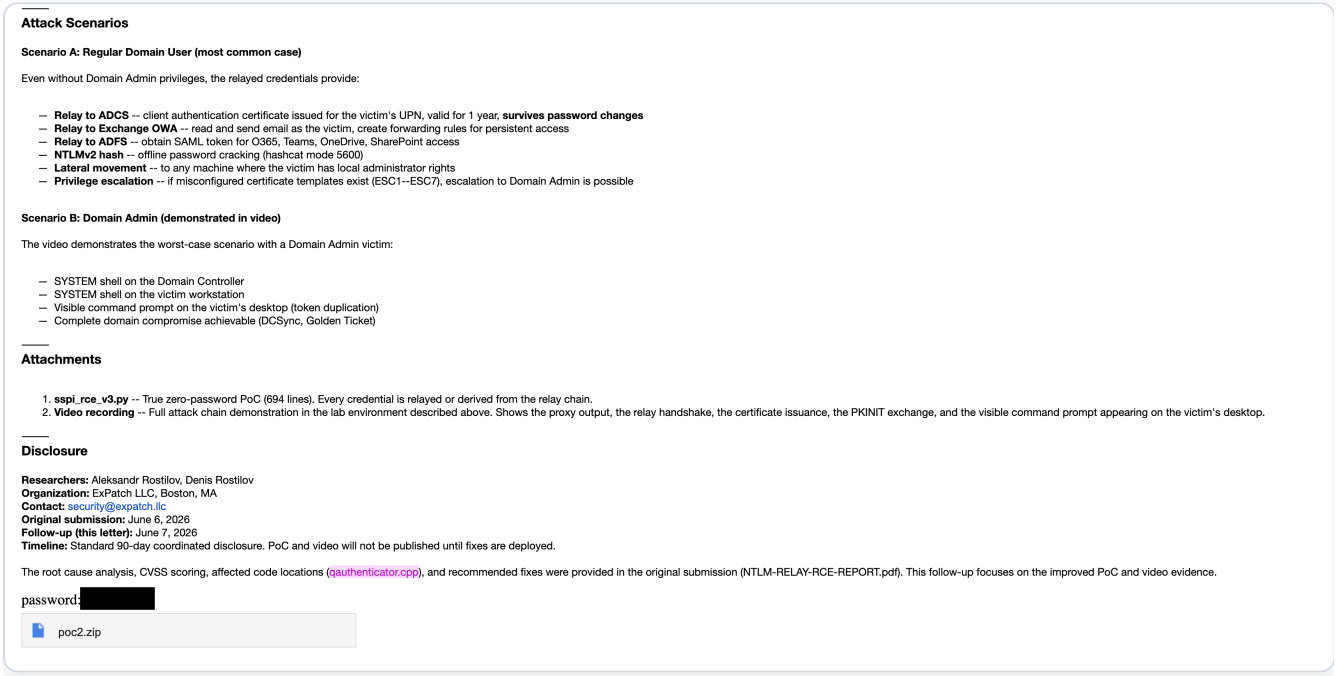
## 7 июня: Telegram получил полностью работоспособный exploit

Первоначальным отчётом раскрытие не ограничилось. Уже 7 июня мы направили Telegram дополнительное письмо с усовершенствованным sspi*rcev*3.py — 694-строчным сквозным эксплойтом, не требующим знания пароля — и видеозаписью полной атаки. Это была не концептуальная демонстрация и не набор разрозненных техник: все необходимые учётные данные автоматически перенаправлялись либо извлекались из результатов relay-цепочки.

В письме были отдельно описаны последствия для обычного доменного пользователя: выпуск клиентского сертификата через AD CS, доступ к Exchange OWA и ADFS, перемещение на компьютеры, где жертва обладала локальными административными правами, а при небезопасной конфигурации

шаблонов сертификатов — повышение привилегий. Для сценария с Domain Admin демонстрация заканчивалась получением NT AUTHORITY\SYSTEM на рабочей станции и контроллере домена; дальнейшие действия могли привести к полной компрометации домена.

Видеозапись последовательно показывала вывод контролируемого прокси, NTLM relay, выпуск сертификата, PKINIT-аутентификацию и появление командной строки на рабочем столе жертвы. Telegram получил не только описание возможного ущерба, но и полностью работоспособную реализацию, воспроизводившую всю цепочку на изолированном стенде против штатного, немодифицированного Telegram Desktop.



---

**Attack Scenarios**

**Scenario A: Regular Domain User (most common case)**

Even without Domain Admin privileges, the relayed credentials provide:

- **Relay to ADCS** -- client authentication certificate issued for the victim's UPN, valid for 1 year, **survives password changes**
- **Relay to Exchange OWA** -- read and send email as the victim, create forwarding rules for persistent access
- **Relay to ADFS** -- obtain SAML token for O365, Teams, OneDrive, SharePoint access
- **NTLMv2 hash** -- offline password cracking (hashcat mode 5600)
- **Lateral movement** -- to any machine where the victim has local administrator rights
- **Privilege escalation** -- if misconfigured certificate templates exist (ESC1--ESC7), escalation to Domain Admin is possible

**Scenario B: Domain Admin (demonstrated in video)**

The video demonstrates the worst-case scenario with a Domain Admin victim:

- SYSTEM shell on the Domain Controller
- SYSTEM shell on the victim workstation
- Visible command prompt on the victim's desktop (token duplication)
- Complete domain compromise achievable (DCSync, Golden Ticket)

---

**Attachments**

1. **sspi_rce_v3.py** -- True zero-password PoC (694 lines). Every credential is relayed or derived from the relay chain.
2. **Video recording** -- Full attack chain demonstration in the lab environment described above. Shows the proxy output, the relay handshake, the certificate issuance, the PKINIT exchange, and the visible command prompt appearing on the victim's desktop.

---

**Disclosure**

**Researchers:** Aleksandr Rostilov, Denis Rostilov
**Organization:** ExPatch LLC, Boston, MA
**Contact:** security@expatch.llc
**Original submission:** June 6, 2026
**Follow-up (this letter):** June 7, 2026
**Timeline:** Standard 90-day coordinated disclosure. PoC and video will not be published until fixes are deployed.

The root cause analysis, CVSS scoring, affected code locations (qauthenticator.cpp), and recommended fixes were provided in the original submission (NTLM-RELAY-RCE-REPORT.pdf). This follow-up focuses on the improved PoC and video evidence.

password: [redacted]

poc2.zip

**Рис. 12.** Дополнительное раскрытие от 7 июня: сценарии эксплуатации, 694-строчный sspircev3.py, видеозапись полной цепочки и прикреплённый архив poc2.zip. В письме также зафиксировано обязательство ExPatch не публиковать PoC и видео до выпуска исправлений.

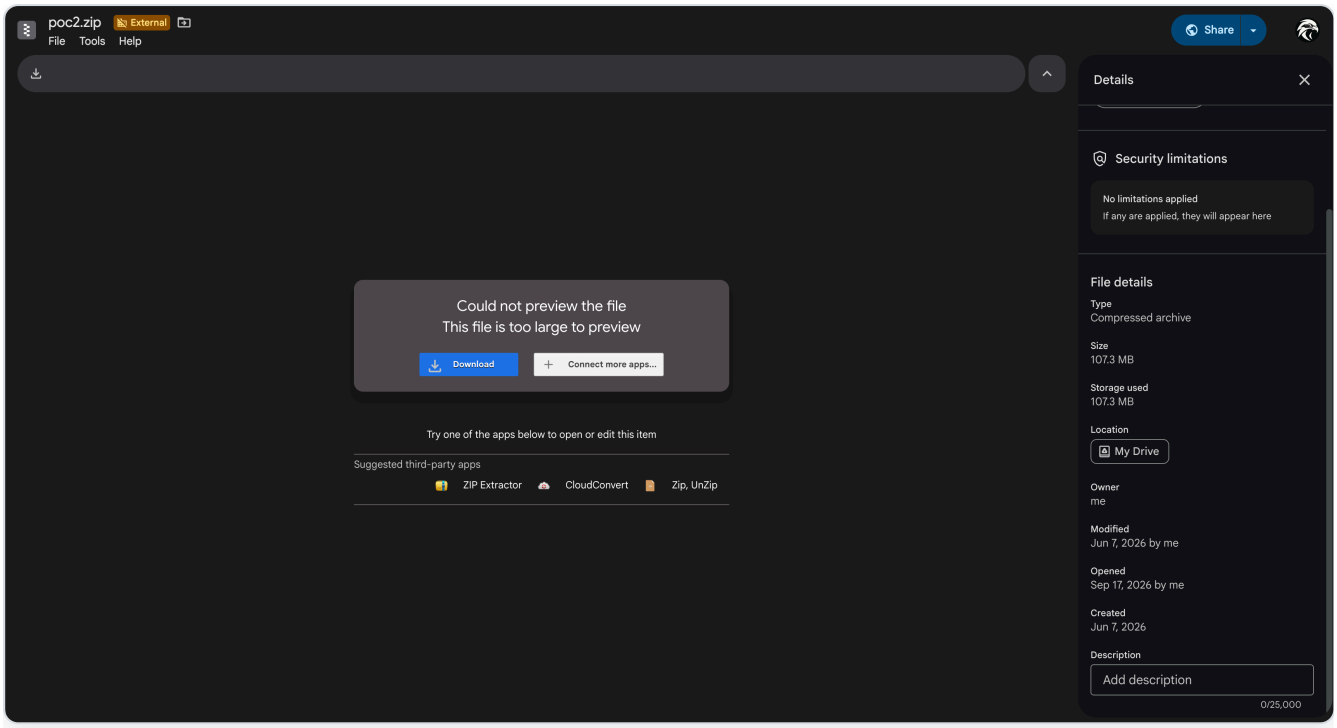



**Рис. 13.** Свойства poc2.zip: Google Drive фиксирует создание и последнее изменение архива 7 июня 2026 года.

## 8 июня: подтверждение расследования и два точечных патча

8 июня в 05:08 Telegram Support подтвердил получение материалов и сообщил:

> “The team is investigating your research, and we will contact you soon.”

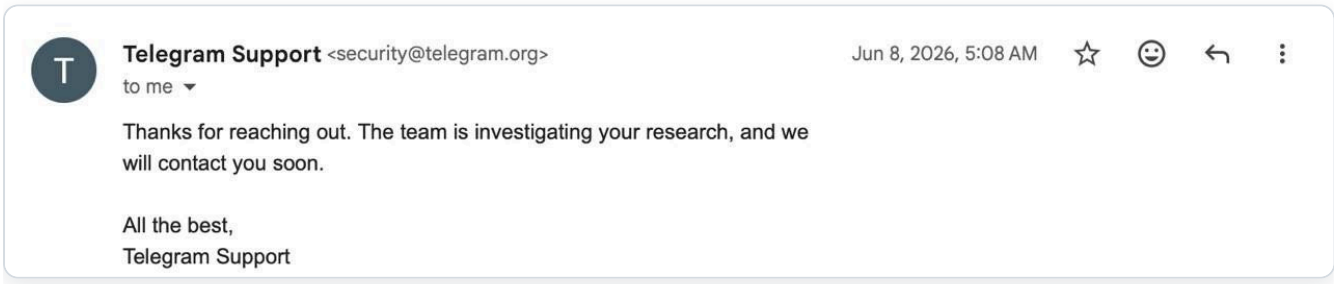



**Рис. 14.** Ответ Telegram от 8 июня 2026 года: команда сообщает, что исследует переданные материалы.

В тот же день публичная история используемого Telegram Desktop репозитория зависимостей датирует появление двух новых коммитов:

- в 13:23:04 UTC — патч для Qt 5 (https://github.com/desktop-app/patches/commit/d92000b4ad84cf8c0fbd161eb32f345ca470da6b);
- в 13:27:01 UTC — патч для Qt 6 (https://github.com/desktop-app/patches/commit/3129a4798e9b69033b5a0a854d4c660b8c719e22).

Автором обоих изменений указан John Preston. По времени Нью-Йорка это 09:23 и 09:27 — примерно через четыре часа после ответа Telegram.

Особенно важно не только время появления изменений, но и их содержание. Патч для Qt 6 был добавлен новым файлом qtbase_6.11.1/0038-disable-proxy-ntlm.patch и одновременно закрывал две независимые проблемы, описанные в нашем отчёте: принудительную NTLM/Negotiate-аутентификацию через прокси и утечку памяти в qExtractServerTime().


**Рис. 15.** Сопоставление отчёта ExPatch от 6 июня с патчем Telegram от 8 июня. Совпадают две независимые находки: запрет NTLM/Negotiate для proxy authentication и исправление обработки MsvAvTimestamp.avLen.

Изменение не ограничивалось общей рекомендацией «усилить NTLM». В нём появилась проверка остатка буфера avLen > end - p, требование avLen == 8, возврат и запись ровно восьми байт, а также отдельная функция qNtlmBufferFits() для проверки границ targetName и targetInfo. Это те же поля, условия и защитные меры, которые были названы в нашем отчёте.

Патч также охватывал оба сетевых пути Qt, использовавшихся Telegram: QHttpNetworkConnection и QHttpSocketEngine. При обнаружении NTLM или Negotiate аутентификатор сбрасывался, соединение завершалось с ProxyAuthenticationRequiredError, а NTLMv2-ответ больше не отправлялся прокси.

## 15–16 июня: предложение награды и подтверждение исправлений

15 июня Telegram поблагодарил нас за отчёт и предложил награду в размере $1000:

> “We would like to award you a bounty of $1000 for your findings regarding NTLM via proxy settings.”

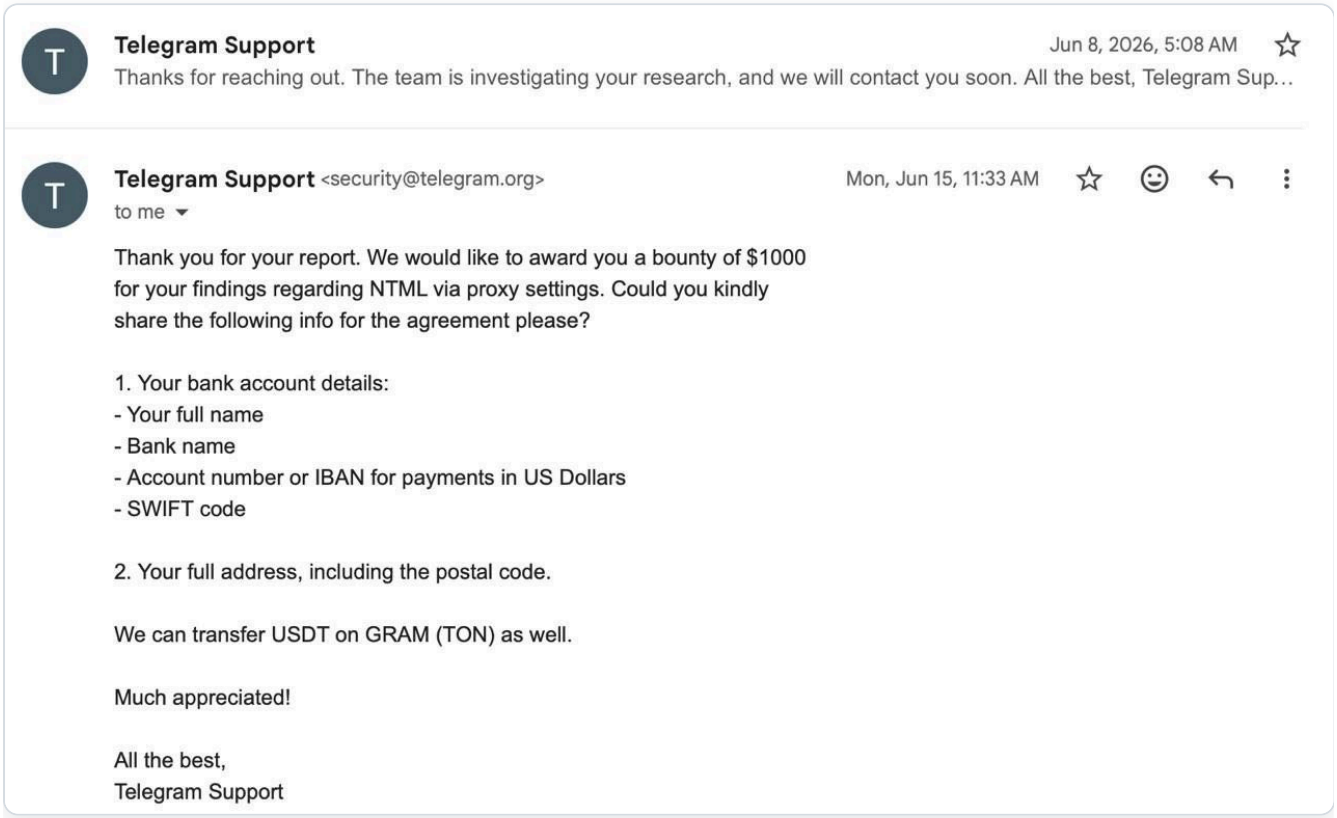



**Рис. 16.** Предложение Telegram от 15 июня 2026 года. Награда сформулирована как относящаяся к находкам, связанным с NTLM через настройки прокси.

К этому моменту патч Telegram уже включал две независимые меры: запрет NTLM/Negotiate для прокси и исправление утечки памяти в qExtractServerTime(). Поэтому формулировка письма не позволяла понять, оценивалась ли вся совокупность переданных сценариев или только proxy-часть исследования.

Деньги не были для нас главным вопросом. Мы попросили направить предложенную сумму на благотворительность и прямо уточнили, является ли решение окончательным для всех описанных уязвимостей либо только для NTLM disclosure через настройки прокси.

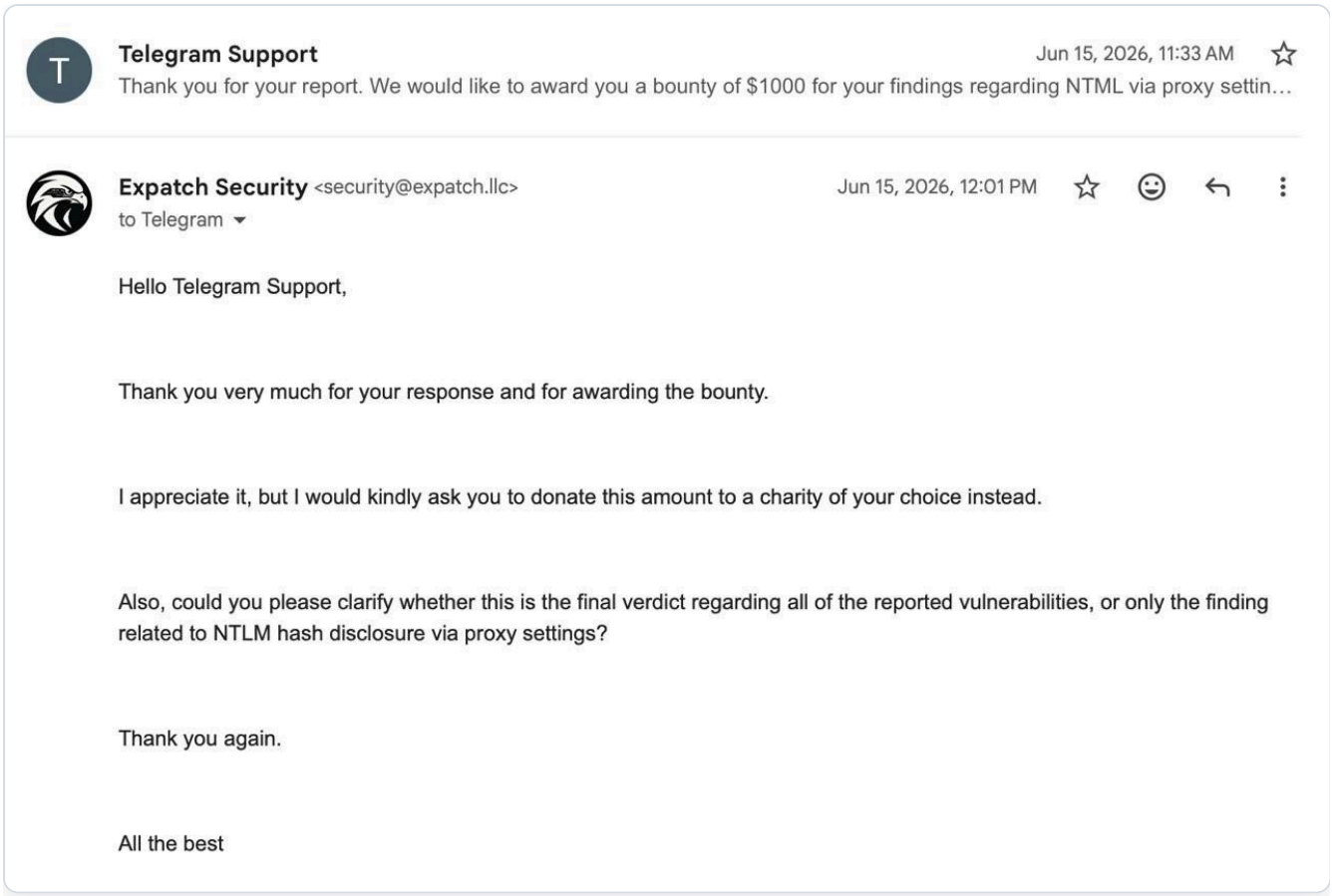
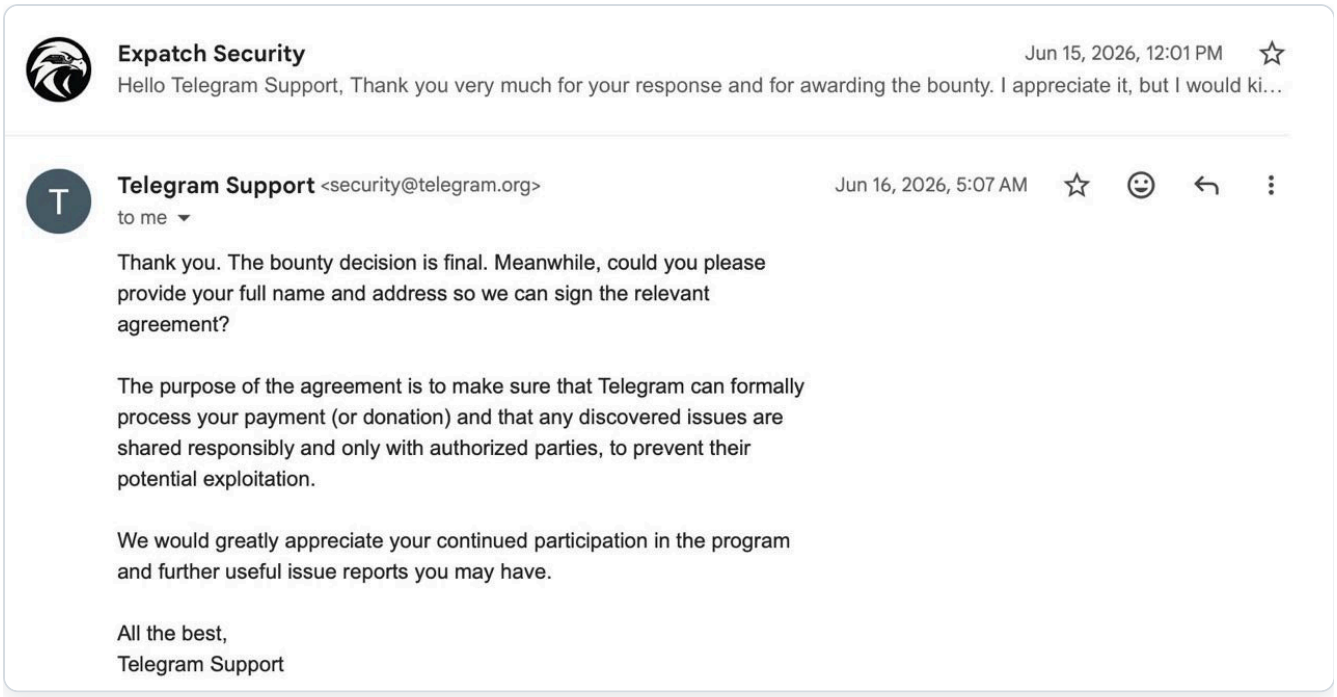



**Рис. 17.** Ответ ExPatch от 15 июня: просьба направить награду на благотворительность и уточнить, какие именно находки охватывает решение.

16 июня в 05:07 Telegram сообщил, что решение о награде окончательное, и напрямую предложил подписать соглашение, фактически выполняющее функцию NDA. По тексту письма оно должно было не только оформить выплату или пожертвование, но и обеспечить, чтобы сведения об обнаруженных проблемах передавались «responsibly and only with authorized parties».



**Expatch Security** Jun 15, 2026, 12:01 PM
Hello Telegram Support, Thank you very much for your response and for awarding the bounty. I appreciate it, but I would ki…

**Telegram Support** <security@telegram.org> Jun 16, 2026, 5:07 AM
to me

Thank you. The bounty decision is final. Meanwhile, could you please provide your full name and address so we can sign the relevant agreement?

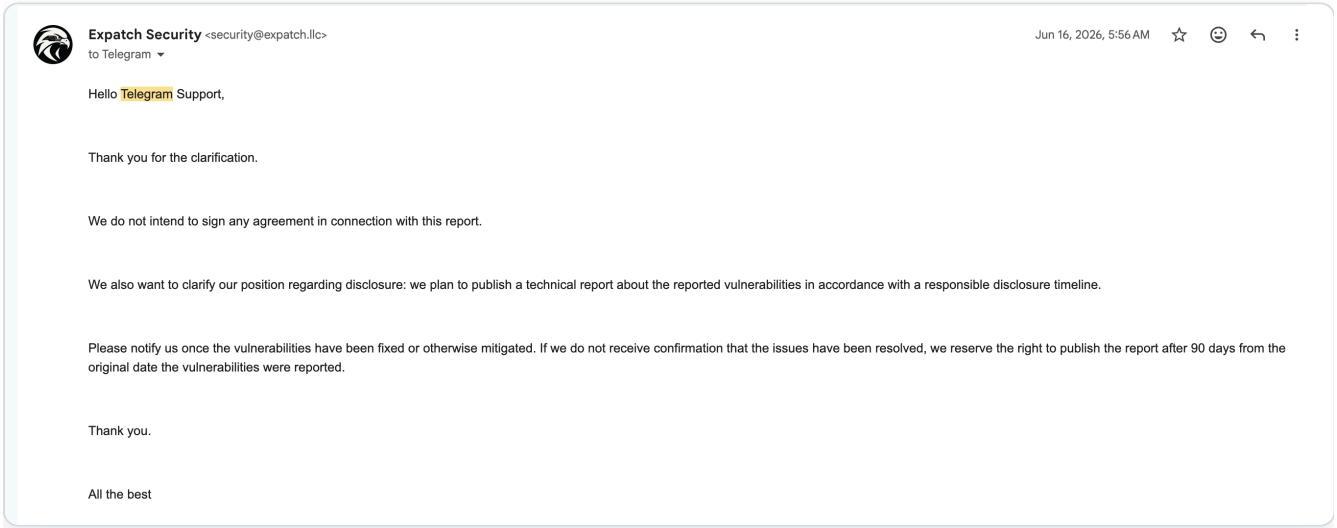
The purpose of the agreement is to make sure that Telegram can formally process your payment (or donation) and that any discovered issues are shared responsibly and only with authorized parties, to prevent their potential exploitation.

We would greatly appreciate your continued participation in the program and further useful issue reports you may have.

All the best,
Telegram Support

**Рис. 18.** Ответ Telegram от 16 июня: предложение подписать соглашение, связывающее оформление выплаты или пожертвования с ограничением распространения сведений только уполномоченными сторонами.

В 05:56 мы сообщили, что не намерены подписывать соглашение, и зафиксировали собственную позицию по responsible disclosure: технический отчёт будет опубликован после устранения уязвимостей либо по истечении 90 дней с даты первоначального обращения. Мы также попросили Telegram уведомить нас, когда проблемы будут исправлены или иным образом нейтрализованы.



**Expatch Security** <security@expatch.llc> Jun 16, 2026, 5:56 AM
to Telegram

Hello Telegram Support,

Thank you for the clarification.

We do not intend to sign any agreement in connection with this report.

We also want to clarify our position regarding disclosure: we plan to publish a technical report about the reported vulnerabilities in accordance with a responsible disclosure timeline.

Please notify us once the vulnerabilities have been fixed or otherwise mitigated. If we do not receive confirmation that the issues have been resolved, we reserve the right to publish the report after 90 days from the original date the vulnerabilities were reported.

Thank you.

All the best

**Рис. 19.** Письмо ExPatch от 16 июня: отказ от соглашения, подтверждение 90-дневного disclosure-срока и просьба сообщить об устранении уязвимостей.

В 08:53 Telegram направил отдельное техническое уточнение:

> “Telegram Desktop v6.9.2 contains fixes for all reported scenarios, with one exception.”

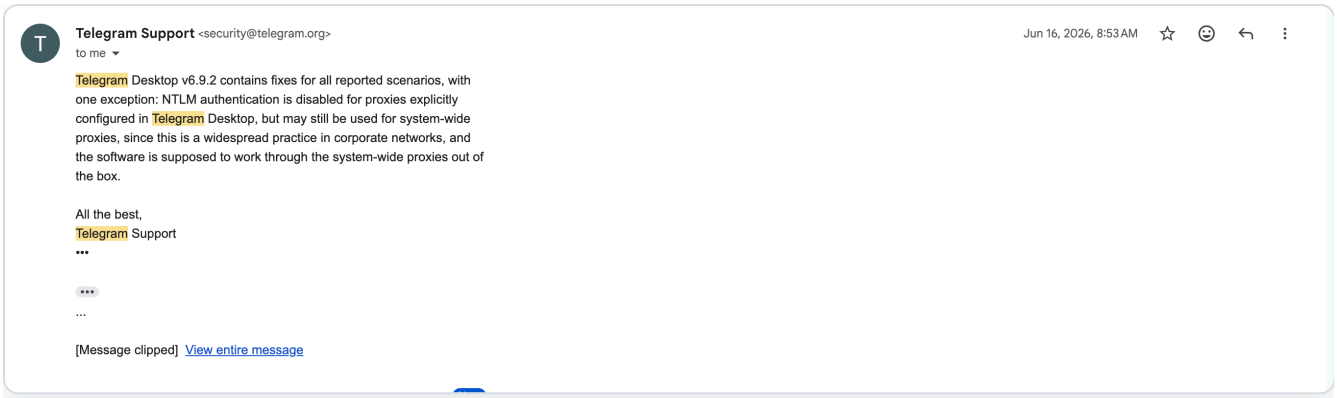



**Рис. 20.** Telegram связывает версию 6.9.2 с исправлением всех переданных сценариев, отдельно указывая исключение для системных прокси.

По словам Telegram, NTLM был отключён для прокси, явно добавленных внутри Telegram Desktop, но оставлен для системных прокси, поскольку такая конфигурация широко используется в корпоративных сетях. Это исключение точно соответствует разделению, которое появилось при доработке патча 10–11 июня: прокси из tg://proxy и настроек приложения блокировались, тогда как системному прокси разрешалось использовать NTLM.

Получив этот ответ, мы решили завершить подготовку отдельного обращения в Qt и скачали актуальную Qt 6.11.2 для контрольного воспроизведения. Однако в этой ветке уязвимые пути уже были закрыты.

Проверив историю Qt после ответа Telegram от 16 июня, мы обнаружили совпадающие upstream-исправления. Telegram не сообщал нам ни о передаче материалов, ни о координации с Qt. ExPatch и авторов исследования исключили из публичной истории исправления.

## Не количество совпадений, а их точность

Мы не будем приписывать себе каждое изменение NTLM-кода, появившееся в Qt в последующие недели. Наш довод не требует длинного списка задач и не строится на общем тематическом сходстве. Он основан на документированной последовательности и соответствии, прослеживаемом до конкретных строк кода.

6 июня Telegram получил наш конфиденциальный отчёт с описанием дефекта в qExtractServerTime(): управляемое сервером поле MsvAvTimestamp.avLen, увеличение QByteArray через timeArray.resize(avLen), неполное чтение входного буфера и последующая отправка непрочитанного содержимого heap внутри NTLMv2-ответа. В отчёте также было предложено проверять avLen по оставшемуся размеру буфера и принимать timestamp только ожидаемой длины — восемь байт.

8 июня история репозитория Telegram датирует появление патча, реализующего эти проверки. В нём появились условия avLen > end - p и avLen != 8, возврат только восьми байтов timestamp и ограничение последующей записи теми же восемью байтами. Одновременно патч закрывал второй описанный нами путь — запрещал NTLM/Negotiate для прокси, явно настроенных внутри Telegram Desktop.

10 июня в Qt был создан QTBUG-147373 (https://qt-project.atlassian.net/browse/QTBUG-147373), зафиксировавший начало upstream-работы по NTLM через два дня после подтверждения Telegram. Связанный коммит: https://github.com/qt/qtbase/commit/6e88e0f70841a5e8f93a126c9a4215c7bc0f53ee. Ещё через два дня, 12 июня, в Gerrit появилось изменение 744448 (https://codereview.qt-project.org/c/qt/qtbase/+/744448): та же функция qExtractServerTime(), то же поле MsvAvTimestamp.avLen, восьмибайтовый timestamp и тесты граничных условий.

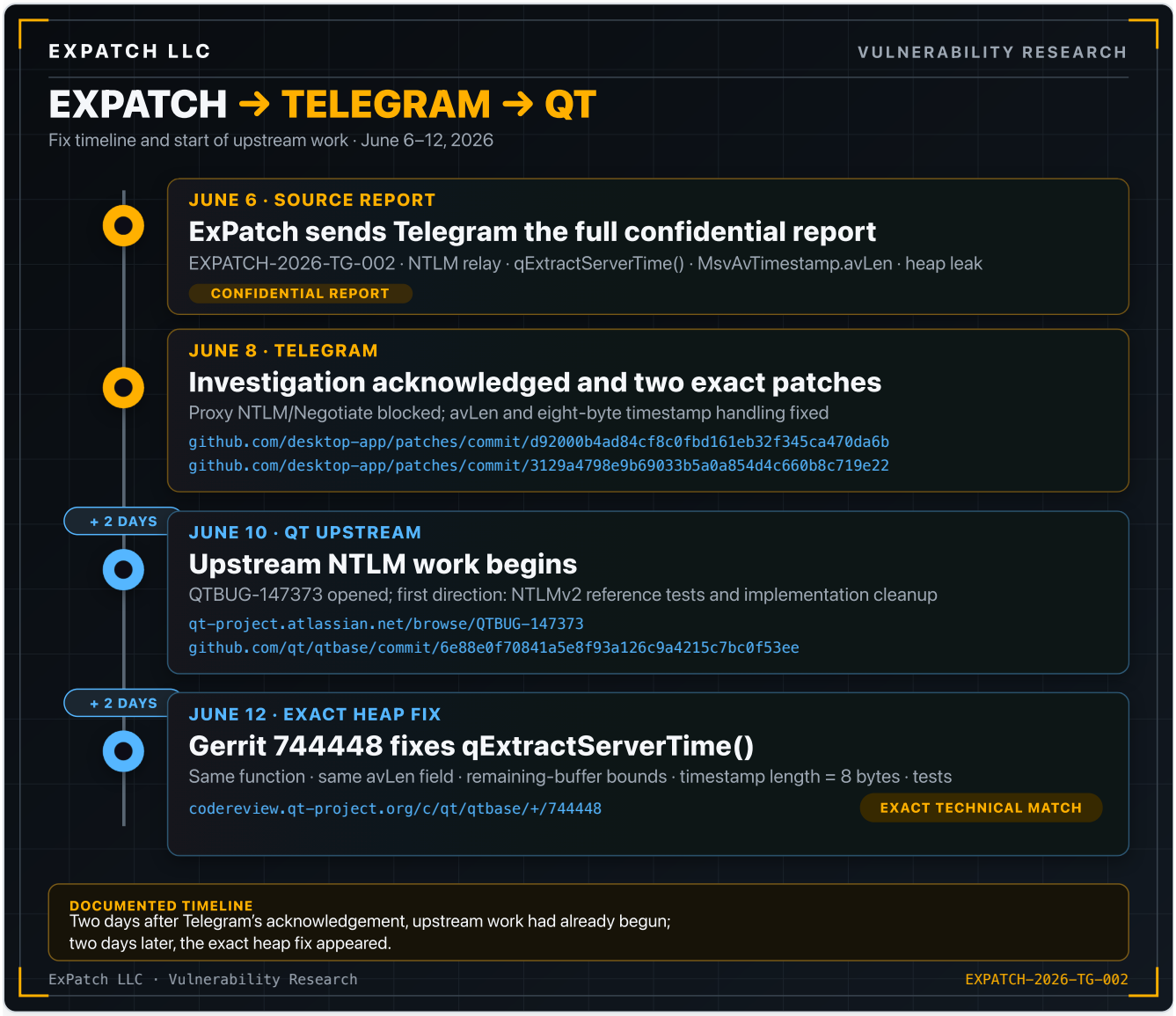



**Рис. 21.** Отчёт ExPatch и соответствующие изменения в Telegram и Qt: хронология 6–12 июня 2026 года.

В патчах Telegram одновременно появились меры против обеих описанных нами проблем: запрет NTLM/Negotiate для прокси, добавленных в приложении, и исправление раскрытия heap-памяти. В Qt затем была исправлена та же ошибка обработки timestamp — с проверкой границ входного буфера и обязательной восьмибайтовой длиной. Соответствие прослеживается на уровне конкретного дефекта, предложенных защитных проверок и поведения исправленного кода.

Эти изменения решали задачи на двух уровнях. Telegram ограничил опасный сценарий внутри своего продукта; исправление Qt устраняло дефект обработки данных в общей библиотеке. Поэтому сведения о работе с upstream были для нас существенны: мы готовили отдельное обращение к разработчикам Qt, а при проверке обнаружили, что соответствующие изменения уже внесены. Пользователи получили защиту, но исследователей не включили в координацию её подготовки.

Ни Telegram, ни Qt публично не объяснили происхождение этого технического соответствия и не указали нас в просмотренных записях. Поэтому вопрос заключается уже не в том, существует ли совпадение, а в том, как результаты переданного нами исследования оказались отражены в upstream-исправлении без уведомления и атрибуции.

## Это не спор о тысяче долларов

Для небольшой исследовательской команды публичная атрибуция — не формальность и не вопрос самолюбия. Это профессиональный результат сорока дней работы, доказательство компетенции и основа доверия, на котором вообще существует coordinated disclosure. Исследователь передаёт вендору неопубликованные материалы первым, предоставляя ему время и техническое преимущество для защиты пользователей. В ответ он вправе ожидать прозрачной координации и сохранения авторства исследования.

Telegram использовал эту фору для подготовки исправлений — и пользователи от этого выиграли. Но после появления соответствующих изменений в Qt нас не уведомили о координации, а ExPatch и имена исследователей не появились в просмотренных публичных записях. Компания получила все преимущества ответственного раскрытия; исследователи не получили даже базовой прозрачности, на которой эта модель держится.

Мы не предлагаем читателю верить нам на слово. Мы публикуем исходный отчёт, его SHA-256 и метаданные, переписку, лабораторные результаты, уязвимый код, патчи и точную последовательность дат. От Telegram и Qt мы ожидаем ответа на том же языке — языке фактов: как результаты нашего конфиденциального исследования оказались отражены в upstream-исправлении и почему их первоисточник остался без атрибуции.

> Благодарим вас за время и внимание к этой работе. Исследование живёт не там, где оно опубликовано, а там, где его читают, проверяют и обсуждают. Именно так отдельная находка становится вкладом в безопасность всех пользователей.

С уважением, Александр Ростилов и Денис Ростилов

> ExPatch Vulnerability Research Team