Исследование безопасности стандарта OPC UA

П.Н. Черемушкин, С.Е. Темников (Kaspersky Lab ICS CERT)

Рассмотрен проект поиска уязвимостей в реализациях стандарта OPC UA. Обращается внимание производителей программного обеспечения для систем промышленной автоматизации и промышленного Internet вещей на типичные проблемы в разработке продуктов с использованием подобных общедоступных технологий. Рассматриваются технические методы, которые могут быть полезными производителям ПО для контроля качества выпускаемых продуктов, а также другим исследователям безопасности программного обеспечения.

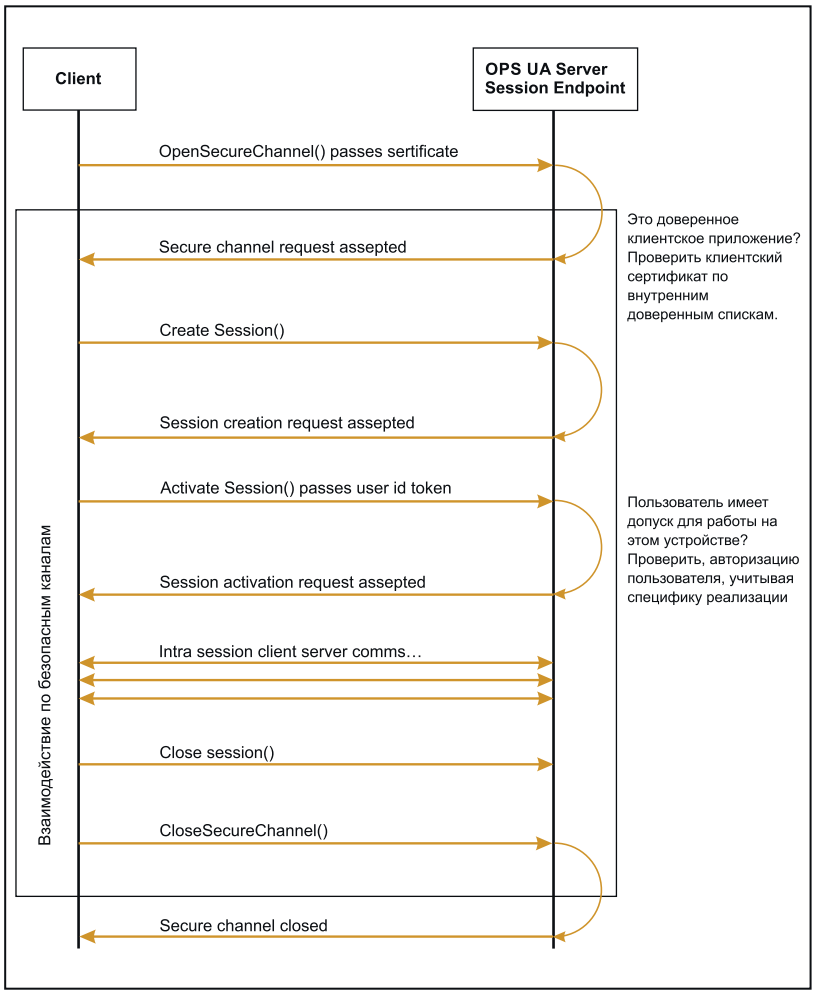
Ключевые слова: стандарт OPC UA, АСУТП, кибербезопасность, уязвимости, фаззинг.

OPC UA как объект исследования

Стандарт IEC 62541 OPC Unified Architecture (OPC UA) разработан в 2006 г. консорциумом OPC Foundation для надежной и, что немаловажно, безопасной передачи данных между различными системами в технологической сети. Стандарт является усовершенствованной версией своего предшественника — OPC, повсеместно применяемого на современных промышленных объектах.

Системы мониторинга и управления, реализованные на продуктах разных производителей, нередко используют несовместимые, часто закрытые, протоколы сетевой коммуникации. Шлюзы/серверы OPC являются связующим звеном между различными АСУТП и системами телеметрии, мониторинга и телеуправления, позволяя унифицировать процессы управления промышленными предприятиями.

Предыдущая версия стандарта, реализованная на технологии Microsoft DCOM, обладала рядом присущих этой технологии существенных ограничений. Для устранения ограничений технологии DCOM и решения некоторых других обнаруженных за время использования протокола OPC проблем консорциум OPC Foundation разработал и выпустил новую версию стандарта[1].



Алгоритм обмена данными по протоколу OPC UA

Благодаря своим новым свойствам и продуманной архитектуре стандарта OPC UA стремительно набирает популярность среди производителей систем автоматизации, шлюзы OPC UA активно внедряются на промышленных предприятиях по всему миру. Протокол становится основой коммуникаций между компонентами систем промышленного Internet вещей и умных городов.

Стандарт OPC UA

Изначально OPC UA разработан для поддержки двух типов передаваемых данных: традиционного (для предыдущих версий стандарта) бинарного формата и SOAP/XML. На сегодняшний день передача данных в формате SOAP/XML считается устаревшей и почти не используется в современных ИТ продуктах и решениях. Перспективы его широкого применения в системах промышленной автоматизации в будущем пока туманны. Поэтому далее речь пойдет о бинарном формате.

Алгоритм обмена данными по протоколу OPC UA выглядит следующим образом (рисунок). Первым сообщением всегда является HELLO (HEL), которое служит маркером начала передачи данных между клиентом и сервером. В ответ сервер посылает клиенту сообщение ACKNOWLEDGE (ACK). После пер-

[1] OPC UA стандарт имеет различные реализации. В данной статье рассматривается только конкретная реализация протокола, разработанная консорциумом OPC Foundation.

вичного обмена сообщениями клиент обычно посылает сообщение OPEN, которое обозначает открытие канала передачи данных с предложенным клиентом методом шифрования данных. В ответ сервер отправляет сообщение OPEN (OPN), которое содержит уникальный ID канала передачи данных, а также показывает, что он согласен на предложенный метод шифрования (или его отсутствие).

Далее, клиент и сервер могут начинать обмен сообщениями MESSAGE (MSG), каждое из которых содержит ID канала передачи данных, тип запроса или ответа, временную метку, массивы передаваемых данных и прочее. В конце сессии передачи данных посылается сообщение CLOSE (CLO), после чего соединение обрывается.

Предпосылки проведения исследований

Команда Kaspersky Lab ICS CERT проводила аудиты безопасности и тесты на проникновение для нескольких промышленных предприятиях. На всех этих предприятиях использовался один и тот же программный продукт для управления технологическим процессом. Заказчиком была поставлена задача в рамках теста на проникновение осуществить поиск уязвимостей в этом программном продукте.

В ходе предварительного обследования оказалось, что часть сетевых сервисов изучаемой системы взаимодействуют по протоколу OPC UA, а большинство исполняемых файлов используют динамическую библиотеку uastack.dll. Таким образом, задача свелась к исследованию безопасности реализации протокола OPC UA.

При поиске бинарных уязвимостей один из самых эффективных методов — фаззинг [1, 2][2]. При исследовании продуктов на Linux-системе использовались методы бинарной инструментации[3] исходного кода и фаззер AFL [3].

Для проведения исследования было решено использовать метод «глупого» фаззера, основанный на мутациях. «Глупый» фаззинг, несмотря на свое название, часто бывает способен существенно повысить вероятность нахождения уязвимостей. Разработка «умного» фаззера для конкретной программы с учетом принципов и алгоритмов ее работы — процесс трудоемкий. А «глупый» фаззер помогает быстро обнаружить тривиальные уязвимости, до которых бывает сложно добраться руками, особенно когда объем кода анализируемого ПО весьма велик. Так было и в рассматриваемом случае.

Архитектура OPC UA Stack делает фаззинг функций напрямую в памяти процесса трудоемкой задачей. Для правильной работы функций, которые требуется проверить на предмет уязвимостей, в процессе фаззинга необходимо передавать правильно сформированные аргументы функций и инициализировать глобальные переменные, которые являются большими по числу полей структурами. Было решено не прибегать к фаззингу функций напрямую из памяти, общаться с изучаемым приложением по сети.

Фаззер работал по следующему алгоритму:

- считать входные последовательности данных;
- преобразовать их псевдослучайным образом;
- послать на вход программе по сети;
- принять ответ от сервера;
- повторить.

Созданный базовый набор мутаций (bitflip, byteflip, арифметические мутации, подставить магическое число, обнулить последовательность данных, подставить длинную последовательность данных) позволил выявить первую уязвимость в uastack.dll. Это была уязвимость типа порчи памяти в результате переполнения на куче (heap corruption), успешная эксплуатация которой может давать злоумышленнику возможность удаленного исполнения кода (RCE), в данном случае — с правами NT AUTHORITY/SYSTEM. Обнаруженная уязвимость заключалась в том, что в функции обработки данных, только что считанных из сокета, неправильно подсчитывался размер данных, которые далее копировались в буфер, созданный на куче.

При внимательном рассмотрении удалось обнаружить, что версия библиотеки uastack.dll, содержащая уязвимость, была скомпилирована разработчиками продукта. По всей видимости, уязвимость была привнесена в код в процессе его модификации. Найти эту уязвимость в версиях библиотеки от OPC Foundation не удалось.

Вторая уязвимость была найдена в приложении, написанном на.NET, которое использовало UA. NET Stack. В ходе анализа трафика приложения с помощью Wireshark[4] было замечено, что в диссекторе у некоторых пакетов есть битовое поле is_xml, в котором в качестве значения стоял нуль. В ходе анализа приложения было обнаружено, что оно использует функцию XmlDocument, которая для.NET версии 4.5 и ранее была уязвима к атаке XXE. То есть замена в битовом поле is_xml нуля на единицу и добавление в тело запроса специальным образом сформирован-

---

[2] Фáззинг (англ. fuzzing) — техника тестирования программного обеспечения, часто автоматическая или полуавтоматическая, заключающая в передаче приложению на вход неправильных, неожиданных или случайных данных.

В зависимости от интеллекта фаззеры бывают глупые и умные:

– глупый (dump) фаззер ничего не знает о структуре файлов. Применительно к сетевым протоколам единственное, что он может сделать — это изменить несколько байтов в исходном пакете и отправить его в надежде, что это может вызвать сбой;

– умный (smart) фаззер имеет некоторое представление о структуре данных, он может воздействовать только на данные, которые, например, отвечают только за размер буфера; или подставлять в поля такие значения, которые заведомо (с учетом известного формата) будут некорректными.

[3] В области программирования под инструментированием понимают возможность отслеживания или установления количественных параметров уровня производительности программного продукта, а также возможность диагностировать ошибки и записывать информацию для отслеживания причин их возникновения.

[4] Wireshark — это известный инструмент для захвата и анализа сетевого трафика.

ного XML-пакета (атака XXE) позволяет получить возможность удаленного чтения файла (out-of-bound file read) с правами NT AUTHORITY/SYSTEM, а при некоторых условиях и возможность удаленного исполнения кода (RCE).

Данное приложение, судя по метаданным, хотя и входило в набор ПО рассматриваемой системы автоматизации, было разработано не вендором, а консорциумом OPC Foundation, и являлось обычным discovery сервером. Это означает, что и другие продукты, которые используют технологию OPC UA от OPC Foundation, могут содержать данный сервер и будут уязвимы к XXE атаке, что делает эту уязвимость гораздо более ценной с точки зрения атакующего.

Так был реализован первый шаг в исследовании, по результатам которого было принято решение продолжить изучение реализации OPC UA от консорциума OPC Foundation, а также продуктов, которые его используют.

Исследование OPC UA

Для выявления уязвимости в реализации протокола OPC UA от консорциума OPC Foundation, необходимо исследовать:

- OPC UA Stack (ANSI C,.NET, JAVA);
- приложения OPC Foundation, которые используют OPC UA Stack (такие как упомянутый выше OPC UA. NET Discovery Server);
- приложения прочих разработчиков ПО, которые используют OPC UA Stack.

Фаззинг UA ANSI C Stack

Разработчики OPC Foundation предоставляют библиотеки, которые являются набором экспортируемых функций, подобных API соответствующих спецификаций. В таких случаях часто сложно определить, является ли обнаруженная потенциальная проблема безопасности на самом деле уязвимостью. Для однозначного ответа на этот вопрос необходимо понимать, как и для чего используется потенциально уязвимая функция, то есть нужен пример программы, использующей данную библиотеку. В рассматриваемом случае было трудно судить об уязвимостях в OPC UA Stack в отрыве от реализаций конечного приложения.

В решении этой проблемы с поиском уязвимостей помог открытый код, размещенный в репозитории OPC Foundation на GitHub, в котором присутствует пример сервера, использующего UA ANSI C Stack. В ходе исследования компонентов системы автоматизации исследователям не так часто удается получить исходный код продуктов. Большинство приложений системы автоматизации — коммерческие продукты, разработанные чаще всего для ОС Windows и выпускаемые с лицензионным соглашением, которое не предполагает открытие исходного кода. В данном случае исследование открытого кода помогло обнаружить ошибки как в самом сервере, так и в библиотеке. Исходный код UA ANSI C Stack пригодился для ручного анализа кода и фаззинга, он также помог понять, добавлена ли вендором новая функциональность в конкретную реализацию UA ANSI C Stack.

UA ANSI C Stack (как и практически все продукты консорциума OPC UA) позиционируется не только как безопасное, но и как кроссплатформенное решение. Это очень помогло в ходе фаззинга, так как удалось собрать UA ANSI C Stack вместе с примером кода сервера, которые разработчики опубликовали в своем GitHub аккаунте, на Linux-системе с бинарной инструментацией исходного кода и провести его фаззинг при помощи AFL.

Для ускорения фаззинга были перегружены функции по работе с сетью так, что они считывали входные данные из локального файла (не из сети), а также скомпилировали нашу программу с AddressSanitizer.

Для формирования исходного набора примеров был использован метод захвата трафика, который идет к сервису от произвольного клиента при помощи tcpdump. Кроме того, в фаззер были добавлены словарь, сформированный исключительно для OPC UA, и специальный механизм мутации, который работал с внутренними структурами OPC UA и изменял их в зависимости от типа

Собрав все усовершенствования фаззера, в течение нескольких минут исследователи получили первое падение программы.

Анализ дампов памяти процесса в момент падения позволил обнаружить в UA ANSI C Stack уязвимость, эксплуатация которой может привести как минимум к DoS.

Фаззинг приложений от OPC Foundation

На предыдущем этапе тестирования рассмотрен фаззинг UA ANSI C Stack и пример приложения от OPC Foundation. В дальнейшем при исследовании готовых продуктов консорциума планировалось избежать повторного тестирования OPC UA Stack и провести фаззинг только отдельных компонентов, написанных поверх стека. Для этого требовалось знание архитектуры OPC UA, а также знание о том, чем приложения, использующие OPC UA Stack, отличаются друг от друга.

Двумя основными функциями в любом приложении, использующем OPC UA Stack, являются функции OpcUa_Endpoint_Create и OpcUa_Endpoint_Open. Первая указывает приложению возможные типы каналов обмена данными между сервером и клиентом, а также список доступных сервисов. Функция OpcUa_Endpoint_Open определяет, из какой сети будет доступен сервис, а также какие режимы шифрования будут на нем доступны.

Список доступных сервисов задается при помощи таблицы сервисов, в которой перечислены структуры данных с информацией непосредственно о каждом сервисе. В каждой такой структуре содержатся данные о типе поддерживаемого запроса, типе ответа, а также две callback-функции, которые будут вызва-

ны в ходе пре- и постобработки запроса (функции преобработки чаще всего являются «заглушками»). В функцию преобработки запросов был помещен код преобразователя, который использует мутированные данные на входе, а на выходе выдает правильно сформированную структуру, соответствующую типу запроса. Таким образом удалось миновать этап старта приложения, запуск event-loop для создания отдельного потока чтения из псевдо-сокета и т.д., что позволило ускорить фаззинг.

В результате работы усовершенствованного таким образом «глупого» фаззера было найдено еще восемь уязвимостей в приложениях от OPC Foundation.

Исследование сторонних приложений, использующих OPC UA Stack

Следующим этапом исследований стало изучение коммерческих продуктов, использующих OPC UA Stack. Из систем автоматизации, встретившихся в процессе проведения тестов на проникновение и исследования состояния защищенности объектов нескольких заказчиков, были выбраны продукты от различных вендоров, включая мировых лидеров этой отрасли. Согласовав работы с заказчиками, мы приступили к исследованию реализации протокола OPC UA в этих продуктах.

На предыдущих этапах при исследовании продуктов на Linux-системе применялся метод бинарной инструментации исходного кода и фаззер AFL. Исследуемые коммерческие продукты, использующие OPC UA Stack, разработаны под ОС Windows, для которой есть аналог фаззера AFL под названием WinAFL. По сути, WinAFL — это портированный на Windows фаззер AFL. Однако ввиду различий между ОС в нем присутствует ряд существенных изменений. Вместо системных вызовов из ядра Linux он использует функции WinAPI, а вместо статической инструментации исходного кода — динамическую инструментацию бинарных файлов DynamoRIO. В сумме эти отличия существенно снижают производительность WinAFL по сравнению с AFL.

Для реализации работы с WinAFL по общей схеме необходимо написать программу, которая будет считывать из специально созданного файла данные и вызывать функцию из исполняемого файла или библиотеки. Тогда WinAFL зациклит этот процесс при помощи бинарной инструментации и будет вызывать эту функцию много раз, получая отклик от запущенной программы и перезапуская функцию с мутированными данными в качестве аргументов. Это позволяет не перезапускать программу каждый раз на новых входных данных, экономя ресурсы системы, так как создание нового процесса в ОС Windows занимает много процессорного времени.

К сожалению, данная схема фаззинга для поставленной задачи не подходила. В силу особенностей асинхронной архитектуры OPC UA Stack обработка данных, получаемых и передаваемых по сети, реализуется в call-back-функциях. Поэтому нельзя для каждого типа запросов выделить одну функцию обработки данных, которой можно было бы передать на вход указатель на буфер с данными и их размер, как это требуется фаззером WinAFL.

Поэтому было принято решение усовершенствовать «глупый» фаззер:

— усовершенствовать механизм мутаций, изменив алгоритм генерации данных с учетом наших знаний о типах передаваемых в OPC UA Stack данных;

— создать набор примеров для каждого из поддерживаемых сервисов (в этом помогла библиотека python-opcua, которая имеет функции взаимодействия практически со всеми возможными сервисами OPC UA);

— при использовании фаззера с динамической бинарной инструментацией для тестирования многопоточных приложений, подобных рассматриваемому, задача поиска новых ветвей в коде приложения — нетривиальна, так как сложно определить, какие же именно данные на входе привели к тому или иному поведению приложения. В данном исследовании эту сложную задачу решать не пришлось, поскольку фаззер общался с приложением по сети, и было возможным однозначно связывать ответ сервера с переданными ему данными (коммуникации происходят в рамках одной сессии). Был реализован алгоритм, определяющий факт обнаружения нового пути выполнения программы по получению нового, ранее не наблюдаемого отклика от сервера.

В результате описанных усовершенствований «глупый» фаззер стал не таким уж и «глупым», и число исполнений в секунду выросло с 1–2 до 70 ед., что является хорошим показателем для сетевого фаззинга. С его помощью были обнаружены еще две новые уязвимости, которые нам не удавалось обнаружить при помощи умного фаззинга.

Результаты

На конец марта 2018 г. итогами исследования стали 17 найденных и закрытых уязвимостей нулевого дня в продуктах OPC Foundation и несколько уязвимостей — в коммерческих приложениях, которые их используют.

Обо всех найденных в процессе исследования уязвимостях было незамедлительно сообщено разработчикам уязвимого ПО.

На протяжении всего исследования специалисты OPC Foundation и представители команд разработки коммерческих продуктов очень быстро реагировали на присылаемую информацию о найденных уязвимостях и своевременно их устраняли.

Большинство ошибок в продуктах сторонних производителей ПО, использующих OPC UA Stack, оказалось связано с тем, что разработчики неправильно использовали функции из предоставляемого OPC Foundation API, реализованного в библиотеке uastack.dll, например, неверно интерпретировали значения полей передаваемых структур данных.

Также было обнаружено, что в нескольких случаях модификация библиотеки uastack.dll разработчиками коммерческого ПО приводила к уязвимости в продукте. Один из примеров — небезопасная реализация функций чтения данных из сокета в коммерческом продукте. Оригинальная реализация этой функции от OPC Foundation при этом ошибки не содержала. Зачем разработчику коммерческого ПО потребовалось модифицировать логику чтения, неизвестно. Однако очевидно, что важности дополнительных проверок, заложенных в реализации от OPC Foundation и обеспечивавших функцию безопасности, разработчик при реализации своего варианта не осознал.

В ходе исследования коммерческого ПО также было обнаружено, что при его создании разработчики берут код примеров использования OPC UA Stack и копируют его в код своих приложений, понадеявшись на то, что OPC Foundation позаботился о безопасности этих фрагментов так же, как и о безопасности кода самой библиотеки. Это их предположение, к сожалению, оказалось неверным.

Эксплуатация некоторых из обнаруженных уязвимостей приводит к DoS и к возможности удаленного исполнения кода. Напомним, что в промышленных системах угроза типа «отказ-в-обслуживании» представляет более существенную опасность, чем в каком-либо другом ПО. Отказ в обслуживании систем телеметрии и телеуправления может приводить к финансовым потерям предприятия, а в некоторых случаях даже к нарушениям и остановке технологического процесса. Теоретически возможна даже порча дорогостоящего оборудования и другой физический ущерб.

Заключение

Тот факт, что консорциум OPC Foundation открывает исходный код своих проектов, безусловно, говорит об открытости и желании консорциума сделать свои продукты более безопасными.

С другой стороны, проведенный анализ показал, что текущая реализация OPC UA Stack не только содержит уязвимости, но и не лишена ряда существенных фундаментальных проблем.

Во-первых, описанные выше ошибки разработчиков коммерческого ПО, использующего OPC UA Stack, позволяют предположить, что OPC UA Stack реализован не совсем очевидным для пользователей образом. Анализ исходного кода от OPC UA Stack это предположение, к сожалению, подтверждает. Текущая реализация протокола изобилует разбросанными по коду вычислениями над указателями, использованием небезопасных структур данных, магических констант, копированием кода проверки параметров из функции в функцию и прочими архаизмами, от которых в современных программных продуктах принято избавляться во многом именно с целью повышения безопасности разработки. Документация кода при этом не отличается подробностью, что повышает вероятность ошибки при его использовании и модификации.

Во-вторых, разработчики OPC UA явно недооценивают степень доверия компаний-разработчиков ПО ко всему коду, предоставляемому консорциумом OPC Foundation. Оставлять уязвимости в коде примеров использования API — совсем неправильно, даже несмотря на то, что примеры использования API не входят в список сертифицируемых продуктов OPC Foundation.

В-третьих, существуют проблемы с контролем качества и сертифицированных продуктов OPC Foundation.

Вероятно, разработчики OPC UA не используют технологию фаззинг-тестирования, наподобие описанной в этой статье, в процессе контроля качества своего продукта. Об этом говорит статистика обнаруженных уязвимостей.

В открытом исходном коде не содержится кода unit- или каких-либо других автоматических тестов, что усложняет тестирование продуктов, использующих OPC UA Stack, в случае модификации кода разработчиками продукта.

Все вышесказанное приводит к довольно неутешительным выводам о том, что разработчики OPC UA, хотя и стараются сделать свой продукт безопасным, тем не менее, пренебрегают современными практиками и технологиями разработки безопасного программного кода.

Текущая реализация OPC UA Stack не только не защищает разработчиков от тривиальных ошибок, но и провоцирует их на совершение ошибок. В современных условиях это неприемлемо для продуктов такого широкого применения, как OPC UA. Тем более это неприемлемо для продуктов, предназначенных для использования в системах промышленной автоматизации.

Список литературы

1. *Ефремов Е.А. Ковалевский А.Е.* Фаззинг. Методы и средства Фаззинга // Международный журнал социальных и гуманитарных наук. 2016. Т. 8. №1. С. 74-76
2. *John Neystadt.* Automated Penetration Testing with White-Box Fuzzing. Microsoft (February 2008). https://msdn.microsoft.com/en-us/library/cc162782.aspx
3. *Michael Sutton, Adam Greene, Pedram Amini.* Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley. 2007. ISBN 0-321-44611-9.

Черемушкин Павел Николаевич *– вирусный аналитик,*
Темников Сергей Евгеньевич *– старший вирусный аналитик Kaspersky Lab ICS CERT.*
Контактный телефон +7 (495) 797-87-00.