АНТИВИРУСНАЯ ЗАЩИТА БИЗНЕС-КЛАССА

Шифратор BitPaymer (FriedEx) создали авторы банковского трояна Dridex

6 февраля 2018 года

Dridex успешно атаковал частных пользователей, компании и финансовые организации на протяжении нескольких лет, став нарицательным категории банковских троянов.

Новое исследование ESET доказывает, что авторы Dridex стоят за другим известным семейством вредоносных программ – сложным шифратором BitPaymer, который детектируется антивирусными продуктами ESET как Win32/Filecoder.FriedEx и Win64/ Filecoder.FriedEx.

Dridex

Банковский троян Dridex был впервые обнаружен в 2014 году и представлял собой относительно простой бот, созданный на базе более старых проектов. Однако авторы быстро превратили его в один из наиболее сложных банковских троянов на рынке. Похоже, что разработка продолжается – новые версии бота с небольшими исправлениями и обновлениями выходят почти еженедельно. Время от времени появляются крупные обновления с новыми функциями или значительными изменениями. Последний апдейт от версии 3 к версии 4 был представлен в начале 2017 года, в нем была внедрена техника инжекта Atom Bombing. Позднее в 2017 году авторы внедрили новый 0-day эксплойт, использующий уязвимость в Microsoft Word, что позволило охватить миллионы жертв.

На момент написания данного поста новейшая версия Dridex 4.80 включает поддержку веб-инжектов в Chrome версии 63. Dridex 4.80 выпущен 14 декабря 2017 года.

Примечание: в прошлом году мы выпустили инструмент, позволяющий идентифицировать вредоносные хуки в популярных веб-браузерах. Инструмент предназначен для помощи в обнаружении потенциальных заражений банковскими троянами, включая Dridex.

FriedEx

Первоначально названный BitPaymer (на базе текста на сайте с требованием выкупа), шифратор открыл в начале июля 2017 года Майкл Гиллеспи. В августе малварь привлекла внимание после успешной атаки на учреждения Национальной службы здравоохранения Шотландии.

FriedEx ориентирован на высокопоставленные цели и компании, а не обычных пользователей, и обычно доставляется посредством RDP-брутфорса. Программа шифрует каждый файл с помощью случайным образом сгенерированного ключа RC4, который затем зашифровывается с использованием жестко закодированного 1024-битного открытого ключа RSA и сохраняется в соответствующем файле .readme_txt.

В декабре 2017 года мы рассмотрели один из образцов FriedEx и практически сразу обнаружили сходства кода с Dridex. Заинтересовавшись находкой, мы провели детальное исследование и выяснили, что FriedEx использует те же методы сокрытия информации о поведении, что и Dridex.

FriedEx «на лету» распознает все вызовы API системы, разыскивая их с помощью хеша, хранит все строки в зашифрованном виде, просматривает ключи и значения реестра с помощью хеша и др. Полученный в результате бинарный файл малозаметен с точки зрения статических свойств; узнать, что делает вредоносное ПО, без более глубокого анализа проблематично.

По этой причине мы провели дальнейший анализ, который выявил дополнительные атрибуты, подтвердившие подозрения – два семейства вредоносных программ созданы одними и теми же разработчиками.

Сходства кода

Dridex loader 2017-10-20

```
  v14 = Buffer::GetSize(v41);
  Buffer::Resize(v41, v14 + 4);
  v15 = Buffer::GetSize(v41);
  v16 = Buffer::GetWritePtrAt(v41, v15 - 4);
  v17 = *(&v37 + v13++);
  *v16 = v17;
}
while ( v13 < 4 );
Reg::GetPathByHash(&v43, v41, HKEY_LOCAL_MACHINE, 0);
if ( v42 )
  sub_CA1D0D(v42, 1);
Buffer::Cleanup(v41);
v18 = Reg::GetValueByHash(&v43, &v29, 0xCD48FA82);
v19 = Reg::QueryDWORD(&v43, *v18);
String::Cleanup(&v29);
v28 = v19;
if ( a4 )
{
  v29 = v19;
  Buffer::Write(v36, &v29, 4);
  v29 = v6;
  Buffer::Write(v36, &v29, 2);
}
else
{
  v29 = v6;
  Buffer::Write(v36, &v29, 2);
  v29 = v28;
  Buffer::Write(v36, &v29, 4);
}
v29 = a3;
Buffer::Write(v36, &v29, 2);
v20 = Buffer::GetSize(v36);
v21 = Buffer::GetWritePtrAt(v36, 0);
Util::HashData(&v37, v21, v20);
v22 = Util::BinToHex(&v37, &v29, 0);
String::ToLowerCase(v22, &v34);
String::Cleanup(&v29);
Buffer::Cleanup(&v37);
if ( v6 )
{
  String::FromCharArray(v5, &v34);
}
else
{
  WString::Init(&v32, 128);
  v28 = v33 >> 1;
  v23 = GetAPIByHash(kernel32, GetComputerNameW);
  if ( v23 )
    v23(v32, &v28);
  Util::WStringToAscii_0(v35, v32);
  v24 = v34;
  v25 = String::AppendChar(v35, '_');
  String::Join(v25, v24);
  String::FromCharArray(v5, v35);
  String::Cleanup(v35);
  String::Cleanup(&v32);
}
```

FriedEx 2017-10-20

```
  v14 = Buffer::GetSize(v41);
  Buffer::Resize(v41, v14 + 4);
  v15 = Buffer::GetSize(v41);
  v16 = Buffer::GetWritePtrAt(v41, v15 - 4);
  v17 = *(&v37 + v13++);
  *v16 = v17;
}
while ( v13 < 4 );
Reg::GetPathByHash(&v43, v41, HKEY_LOCAL_MACHINE, 0);
if ( v42 )
  sub_2A1093(v42, 1);
Buffer::Cleanup(v41);
v18 = Reg::GetValueByHash(&v43, &v29, 0xCD48FA82);
v19 = Reg::QueryDWORD(&v43, *v18);
String::Cleanup(&v29);
v28 = v19;
if ( a4 )
{
  v29 = v19;
  Buffer::Write(v36, &v29, 4);
  v29 = v6;
  Buffer::Write(v36, &v29, 2);
}
else
{
  v29 = v6;
  Buffer::Write(v36, &v29, 2);
  v29 = v28;
  Buffer::Write(v36, &v29, 4);
}
v29 = a3;
Buffer::Write(v36, &v29, 2);
v20 = Buffer::GetSize(v36);
v21 = Buffer::GetWritePtrAt(v36, 0);
Util::HashData(&v37, v21, v20);
v22 = Util::BinToHex(&v37, &v29, 0);
String::ToLowerCase(v22, &v34);
String::Cleanup(&v29);
Buffer::Cleanup(&v37);
if ( v6 )
{
  String::FromCharArray(v5, &v34);
}
else
{
  WString::Init(&v32, 128);
  v28 = v33 >> 1;
  v23 = GetAPIByHash(kernel32, GetComputerNameW);
  if ( v23 )
    v23(v32, &v28);
  Util::WStringToAscii_0(v35, v32);
  v24 = v34;
  v25 = String::AppendChar(v35, '_');
  String::Join(v25, v24);
  String::FromCharArray(v5, v35);
  String::Cleanup(v35);
  String::Cleanup(&v32);
}
```

Рисунок 1. Сравнение функции GetUserID в образцах Dridex и FriedEx – найди десять отличий

На рисунке 1 представлена часть функции, используемой для генерации UserID, который можно найти во всех бинарных файлах Dridex (загрузчики и модули бота). Функция, специфичная для Dridex, используется и в бинарных файлах FriedEx. Задача функции та же – генерировать строку из нескольких атрибутов машины жертвы, которая служит уникальным идентификатором – в ботнете, если речь о Dridex и для шифратора в случае с FriedEx.

Такое сходство с Dridex присутствует во всех бинарных файлах FriedEx. С образцом Dridex не совпадает лишь несколько функций, как правило связанных с конкретными возможностями шифратора (цикл шифрования и создания файлов с сообщением о выкупе).

Dridex loader 2017-10-20

Function name	Segment
Core__GetUserIDInternal	.text
sub_CA5191	.text
sub_CA51A7	.text
sub_CA53F5	.text
Buffer__Init	.text
sub_CA5457	.text
Buffer__Cleanup	.text
Buffer__Copy	.text
sub_CA54C8	.text
sub_CA5508	.text
Buffer__GetWritePtrAt	.text
Buffer__Write	.text
Util__BinToHex	.text
sub_CA5616	.text
sub_CA562B	.text
sub_CA563A	.text
sub_CA5642	.text
Buffer__Write_0	.text
sub_CA56C0	.text
sub_CA56DA	.text
sub_CA5757	.text
Crypto__GenRandomData	.text
Buffer__InitInternal	.text
sub_CA581B	.text
Buffer__Resize	.text
Buffer__GetSize	.text
String__FromCharArray	.text
String__Init	.text
sub_CA5898	.text
sub_CA58B5	.text
WString__Init	.text

FriedEx 2017-10-20

Function name	Segment
Core__GetUserIDInternal	.text
sub_2A56A2	.text
sub_2A58F0	.text
Buffer__Init	.text
sub_2A5952	.text
Buffer__Cleanup	.text
Buffer__Copy	.text
Buffer__GetWritePtrAt	.text
Buffer__Write	.text
Util__BinToHex	.text
sub_2A5ABF	.text
sub_2A5AD4	.text
Buffer__Write_0	.text
sub_2A5CDC	.text
Crypto__GenRandomData	.text
Buffer__InitInternal	.text
sub_2A5DA0	.text
Buffer__Resize	.text
Buffer__GetSize	.text
MemsetWrapper_0	.text
String__FromCharArray	.text
String__Init	.text
sub_2A5E1D	.text
sub_2A5E3A	.text
WString__Init	.text

Рисунок 2. Сравнение порядка функций в образцах Dridex и FriedEx. Функции, отсутствующие в другом образце, выделены цветом

Еще одна общая черта – последовательность функций в бинарных файлах, которая возникает, когда одна и та же кодовая база или статическая библиотека используется в нескольких проектах. Как мы можем видеть на рисунке 2, в образце FriedEx отсутствуют некоторые функции, присутствующие в образце Dridex, и наоборот, но порядок остается неизменным.

Примечание: автоматические сгенерированные пары имен функций, основанные на адресах в коде (sub_CA5191 и sub_2A56A2 и т.п.), очевидно не совпадают, зато совпадает код, на который они ссылаются.

Стоит упомянуть, что и Dridex, и FriedEx используют один и тот же вредоносный упаковщик. Однако этот упаковщик в настоящее время крайне популярен, что, вероятно, связано с его эффективностью в предотвращении детектирования и осложнении анализа. Он используется в других семействах вредоносных программ, включая QBot, Emotet и Ursnif, поэтому мы не считаем его наличие убедительным доказательством.

Пути PDB

При создании исполняемого файла Windows компоновщик может включать путь PDB (Program Database), указывающий на файл, который содержит символы отладки, помогающие разработчику исправлять ошибки и идентифицировать сбои. Фактически PDB-файл почти никогда не присутствует во вредоносных программах, поскольку представляет собой отдельный файл, не попадающий в дистрибутив.

Путь может предоставлять ценную информацию, поскольку PDB-файлы расположены по умолчанию в том же каталоге, что и скомпилированный исполняемый файл, и обычно имеют то же базовое имя, за которым следует расширение .pdb. Логично, что путь PDB обычно не включают во вредоносные программы, поскольку авторы не хотят раскрывать какую-либо информацию. К счастью, в некоторых образцах Dridex и FriedEx путь PDB есть.

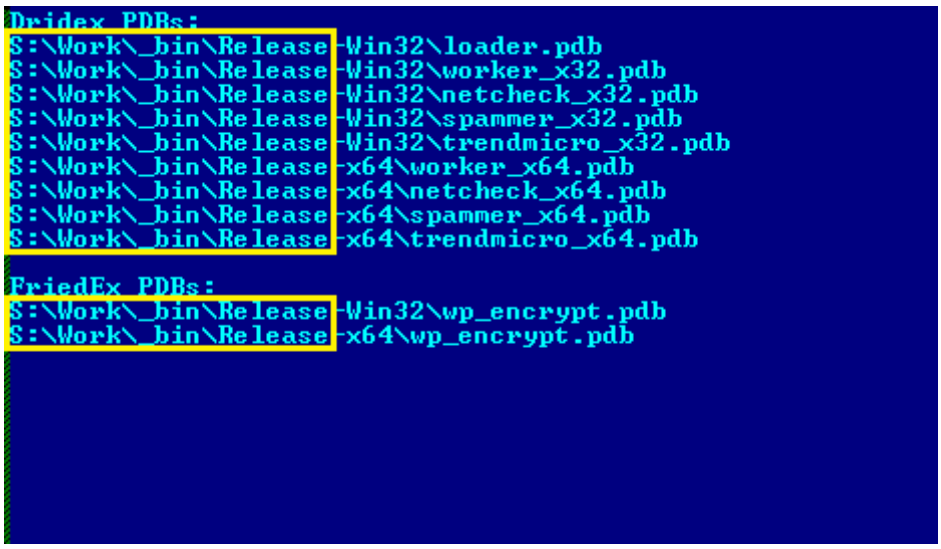

Рисунок 3. Все пути PDB, найденные в проектах Dridex и FriedEx

Как можно видеть на рисунке 3, бинарные файлы обоих проектов собраны в одном и том же каталоге. На базе поиска в метаданных имеющихся образцов вредоносного ПО мы пришли к выводу, что путь S:\Work_bin\ уникален для проектов Dridex и FriedEx.

Временные метки

У нас есть несколько случаев обнаружения Dridex и FriedEx с одной и той же датой компиляции. Это могло быть совпадением, но после более пристального изучения мы исключили такую версию.

Мало того, что компиляции с одинаковыми датами различаются на несколько минут (можно предположить, что авторы Dridex одновременно компилировали оба проекта), но случайно сгенерированные константы в этих образцах также идентичны. Константы изменяются в каждой компиляции как форма полиморфизма, чтобы затруднить анализ и помочь избежать детектирования. Это может быть полностью рандомизировано в каждой компиляции или на основе какой-либо переменной, такой как текущая дата.

Dridex loader 2017-10-17

```
int __fastcall GetAPIByHash(LIB_HASHES a1, FUNC_HASHES a2)
{
  int v2; // edi
  void *v3; // esi
  int result; // eax
  int v5; // eax

  v2 = a2;
  v3 = a1;
  result = GetAPIByHash_Fast(a2);
  if ( !result )
  {
    if ( v3 != 0x634B12BD
      && ((v5 = FindDLLByHash(v3)) != 0
       || LoadDLLByHash(v3)
       && (v5 = FindDLLByHash(v3)) != 0) )
    {
      result = GetProcByHash(v5, v2);
    }
    else
    {
      result = 0;
    }
  }
  return result;
}
Machine                         Intel386
Tue Oct 17 17:50:02 2017
Magic optional header           0108
```

Dridex loader 2017-10-20

```
int __fastcall GetAPIByHash(LIB_HASHES a1, FUNC_HASHES a2)
{
  FUNC_HASHES v2; // edi
  LIB_HASHES v3; // esi
  int result; // eax
  int v5; // eax

  v2 = a2;
  v3 = a1;
  result = GetAPIByHash_Fast(a2);
  if ( !result )
  {
    if ( v3 != 0x26A34D30
      && ((v5 = FindDLLByHash(v3)) != 0
       || LoadDLLByHash(v3)
       && (v5 = FindDLLByHash(v3)) != 0) )
    {
      result = GetProcByHash(v5, v2);
    }
    else
    {
      result = 0;
    }
  }
  return result;
}
Machine                         Intel386
Fri Oct 20 16:10:29 2017
Magic optional header           0108
```

Рисунок 4. Функция GetAPIByHash в образцах Dridex с разницей времени компиляции в три дня. Выделенные константы различаются

На рисунке 4 мы сравниваем два образца загрузчика Dridex с трехдневной разницей времени компиляции. Загрузчики почти идентичны, единственное различие – жестко закодированные данные, такие как ключи шифрования и IP-адреса C&C-серверов. При этом константы различны, и поэтому все хеши основаны на них.

С другой стороны, на рисунке 5 можно видеть сравнение загрузчиков Dridex и FriedEx, скомпилированных в один день (разница временных меток в две минуты). Константы одинаковы, это указывает, что оба образца были созданы в ходе одной сессии компиляции.

Dridex loader 2017-09-07

```
int __fastcall GetAPIByHash(LIB_HASHES a1, FUNC_HASHES a2)
{
  FUNC_HASHES v2; // edi
  LIB_HASHES v3; // esi
  int result; // eax
  int v5; // eax

  v2 = a2;
  v3 = a1;
  result = GetAPIByHash_Fast(a2);
  if ( !result )
  {
    if ( v3 != 0xA8A28E83
      && ((v5 = FindDLLByHash(v3)) != 0
       || LoadDLLByHash(v3)
       && (v5 = FindDLLByHash(v3)) != 0) )
    {
      result = GetProcByHash(v5, v2);
    }
    else
    {
      result = 0;
    }
  }
  return result;
}
Machine                         Intel386
Thu Sep 07 17:57:41 2017
Magic optional header           0108
```

FriedEx 2017-09-07

```
int __fastcall GetAPIByHash(LIB_HASHES a1, FUNC_HASHES a2)
{
  FUNC_HASHES v2; // edi
  LIB_HASHES v3; // esi
  int result; // eax
  int v5; // eax

  v2 = a2;
  v3 = a1;
  result = GetAPIByHash_Fast(a2);
  if ( !result )
  {
    if ( v3 != 0xA8A28E83
      && ((v5 = FindDLLByHash(v3)) != 0
       || LoadDLLByHash(v3)
       && (v5 = FindDLLByHash(v3)) != 0) )
    {
      result = GetProcByHash(v5, v2);
    }
    else
    {
      result = 0;
    }
  }
  return result;
}
Machine                         Intel386
Thu Sep 07 17:59:17 2017
Magic optional header           0108
```

Рисунок 5. Функция GetAPIByHash в бинарных файлах Dridex и FriedEx, скомпилированная в один день. Выделенные константы идентичны в обоих образцах

Информация компилятора

Информация компилятора подтверждает доказательства, перечисленные ранее – бинарные файлы Dridex и FriedEx скомпилированы в Visual Studio 2015. Это подтверждает версия компоновщика, найденная в заголовке PE, и данные Rich Header.

Dridex loader 2017-10-20

Internal name	Compiler version	Count
prodidUtc1900_C	Visual Studio 2015 (14.00) build 24215	1
prodidImplib1100	Visual Studio 2012 (11.00) build 65501	3
prodidImport0	Imports build 0	2
prodidUtc1900_CPP	Visual Studio 2015 (14.00) build 24215	26
prodidLinker1400	Visual Studio 2015 (14.00) build 24215	1

FriedEx 2017-10-20

Internal name	Compiler version	Count
prodidUtc1900_C	Visual Studio 2015 (14.00) build 24215	1
prodidImplib900	Visual Studio 2008 (09.00) build 30729	5
prodidImport0	Imports build 0	4
prodidUtc1900_CPP	Visual Studio 2015 (14.00) build 24215	21
prodidLinker1400	Visual Studio 2015 (14.00) build 24215	1

Рисунок 6. Данные Rich header в образцах Dridex и FriedEx

Помимо очевидного сходства с Dridex, мы столкнулись с ранее не задокументированной 64-битной версией шифратора. Поскольку обычная 32-битная версия может быть ориентирована на системы x86 и x64, этот образец представляется довольно интересным.

Вывод

С учетом вышеперечисленных доказательств, мы считаем, что FriedEx – работа создателей Dridex. Это открытие дает нам более полную картину деятельности кибергруппы – мы выяснили, что хакеры сохраняют высокую активность и не только постоянно обновляют банковский троян (поддержка веб-инжекта для последних версий Chrome или внедрение новых функций, включая Atom Bombing), но и следуют за последними трендами, создав собственный шифратор.

Сложно прогнозировать будущее, но мы считаем, что группа Dridex не собирается сворачивать деятельность в ближайшее время, продолжит поддержку старого проекта и, возможно, расширит портфолио с помощью новых образцов.

На протяжении длительного времени группа Dridex считалась «актером одной роли», сосредоточенной на своем банковском трояне. Сегодня мы выяснили, что это не так; хакеры легко адаптируются к новым трендам и создают другие разновидности вредоносного ПО, которые могут конкурировать с наиболее продвинутыми в своей категории.

Индикаторы компрометации

Win32/Dridex.BE C70BD77A5415B5DCF66B7095B22A0DEE2DDA95A0
Win64/FriedEx.A CF1038C9AED9239B6A54EFF17EB61CAB2EE12141
Win32/FriedEx.A 8AE1C1869C42DAA035032341804AEFC3E7F3CAF1