SafeBreach

ABUSING TEXT EDITORS VIA THIRD-PARTY PLUGINS

A SafeBreach Labs research by

Dor Azouri, Security Researcher, SafeBreach

March 2018

EXECUTIVE SUMMARY

Software third-party extensibility mechanisms have proven to be fertile ground for attacks many times in the past. Following recent attacks of such nature, we started examining 6 popular extensible text editors for unix systems, checking their defenses against similar potential abuses.

This research document outlines how some of them can be abused locally for privilege escalation. Mitigating this potential breach can be done in several ways; we provide OSSEC rules for your own convenience.

While there are other known techniques to escalate privileges, specific restrictive circumstances might make this one the preferred choice.

OVERVIEW

Advanced editors give you more features, ease of use and most importantly for our scope - they offer extensibility. Cutting to the chase, extensibility means running some third-party bits of code. That third party may be yourself, or a developer that made his extension public (published in a web store, git repo...).

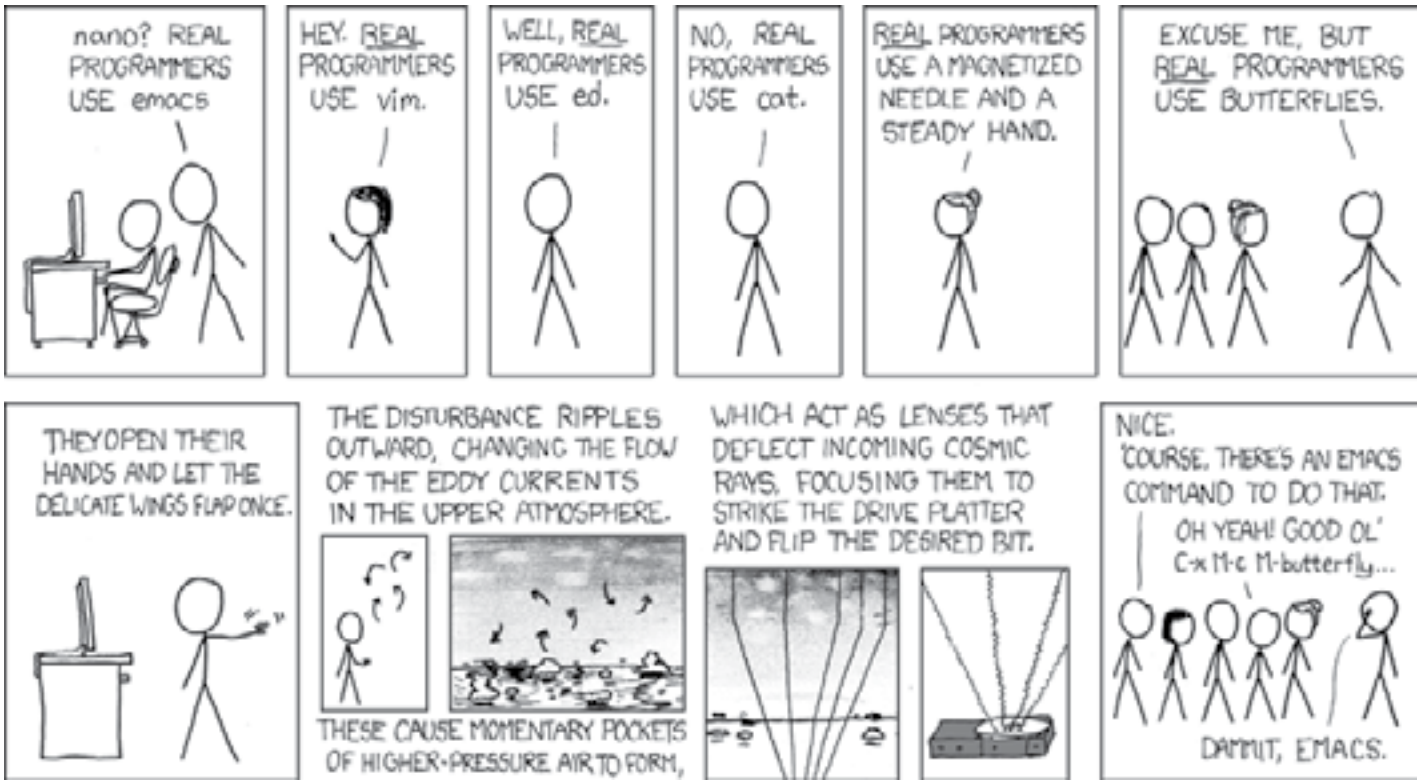
Technical users will occasionally need to edit root-owned files, and for that purpose they will open their editor with elevated privileges, using “sudo”. There are many valid reasons to elevate the privileges of an editor.

We examined several popular editors for unix environments. Our research shows how extensible text editors can be used locally as another way to gain privilege escalation on the machine. We will demonstrate how our method succeeds regardless of the file being opened in the editor, so even limitations commonly applied on sudo commands might not protect from it.

The set of editors that were put to the test include some of the most highly ranked ones (according to http://lifehacker.com/five-best-text-editors-1564907215).

1. Sublime
2. Vim
3. Emacs
4. Gedit
5. pico/nano

Let us detail the findings and describe the ways these editors can be abused.



"Real programmers set the universal constants at the start such that the universe evolves to contain the disk with the data they want."

AS A PRIVILEGE ESCALATION METHOD

What we found about most of the applications that we examined is that when it comes to loading plugins, their separation of the two modes-- regular and elevated--is not complete. Their folder permissions integrity is not kept well and that opens the door for an attacker with regular user permissions to get elevated execution of arbitrary code.

Imagine a starting point where an attacker has the ability to run code, not elevated. The user that he runs under is a sudoer (Linux), but running without elevated status. All he or she has to do is write a malicious plugin to the user folder of the editor that's in use, and wait for the editor to be invoked in elevated status, where the user will enter his root password. Depending on the user profile, the attacker might only need to wait for hours. In some cases he may wait forever, but there are plenty of situations that require users to open files using sudo.

THE ATTACK

Assumptions

For the privilege escalation to work, we assume the following:

- We have regular (not elevated) execution using a sudoer user (a user that's in /etc/sudoers)
- The user occasionally invokes the editor in elevated status - common amongst linux users, especially when the machine is a server.
- The initial user the attacker takes use of has no malicious intentions. He or she has no intentions to exploit his or her sudo permissions to hack or break. If he or she was an insider threat, then of course this method is redundant.
- Alternatively, the user may have malicious intentions, but the system administrator has restricted his sudo privileges to only specific commands and files.

DETAILED DESCRIPTION

All code pieces and other content described below can be found in the SafeBreach-Labs/blog-snippets GitHub repository.

Sublime

Tested on:
Sublime Version: 3.0 Build 3143
Ubuntu 14.04.5, Kernel 4.13.0

Adding a Plugin

A Sublime extension can come in one of two forms:

- An installed package (usually installed using Package Control)
- A stand-alone Python plugin file

The first form is manageable by Package Control, and visible to it. It should typically be found in:

```
/opt/sublime_text
```

as a sublime-package file.

The second form is no more than a Python file placed in:

```
~/.config/sublime-text-3/Packages/User
```

Then if you want your plugin to handle editor events, you should implement your own subclass of sublime_plugin. EventListener. You can then override some event callbacks, the first choice being on_activated_async. This specific event is "Called when a view gains input focus. Runs in a separate thread, and does not block the application". Thus, it is perfect for a plugin that aims to get maximum runtime occasions. This callback will be called every time a user switches tabs, or at first, just when sublime starts. Other events available in the API can be found here.

All that is left to do is for the attackers to implement their malicious intentions with the whole Python environment as their arsenal[1].

VIM

Tested on:
Vim Versions: 7.4 & 8.0
Ubuntu 14.04.5, Kernel 4.13.0

In order to duplicate the method to vim, we need to locate the relevant plugin locations. We also need to know how to load a plugin on startup. The main idea is the same as was previously shown on Sublime:

1. Build your desired python script.
2. Place it in ~/.vim/plugin/.
   If the directory is missing, create it within the directory tree.
3. Create a new *.vim file in the same folder. This file is the actual vim plugin, and it will be loaded on startup along with the other plugins. In our case, we want to demonstrate how to use the python script so the file should look like this (VimScript):

```
if !has('python') && !has('python3')
      finish
endif

function! RootWrite()
      If has('python')
            pyfile ~/.vim/plugin/write_root_file.py
      else
            py3file ~/.vim/plugin/write_root_file.py
      endif
endfunc

call RootWrite()
```

4. The write_root_file.py script in our test is trying to write a file to a protected path, that only root has write access to:

```
try:
      f = open("/stub.file", "wb")
      f.close()
except IOError as e:
      pass
```

[1] A slight "limitation" comes from the fact that the plugins are run in the shipped Python3.3 environment, which excludes some standard modules (Python3.3.zip in the installation folder). But due to the nature of Python, this is not a real limitation - just include the code you want in text/import an external module.

Emacs

Tested on:
Emacs Versions: 25.3.2 & 23
Ubuntu 14.04.5, Kernel 4.13.0
Emacs 24 on Ubuntu 16.04

Emacs executes its init file when it loads, and that's where a user can add key bindings, define functions, and call external commands. It contains personal EmacsLisp code that will be executed when Emacs starts. This file is located in one of the following locations:

- For GnuEmacs, it is ~/.emacs or _emacs or ~/.emacs.d/init.el.
- For XEmacs, it is ~/.xemacs or ~/.xemacs/init.el.
- For AquamacsEmacs, it is ~/.emacs or ~/Library/Preferences/Aquamacs Emacs/Preferences.el

All you have to do is add this ELisp line of code to the init file. It will execute the command "touch /stub.file", when "~/emacs.d/" is the working directory.

```
(let ((default-directory "~/.emacs.d/")) (shell-command "touch /stub.file"))
```

And the privilege escalation objective is possible here as well, because surprisingly, this init file can be edited without root permissions.

Gedit

Tested on:
gedit Version: 3.10.4
Ubuntu 14.04.5, Kernel 4.13.0

Installing a Plugin
Gedit plugins can be installed by either a package manager or locally from a source file. The ones that come from a package manager are located here:

```
    /usr/lib/x86_64-linux-gnu/gedit/plugins/
```

While the local ones should be placed here (mkdirs if plugin folder does not exist):

```
    ~/.local/share/gedit/plugins/
```

Enabling a Plugin

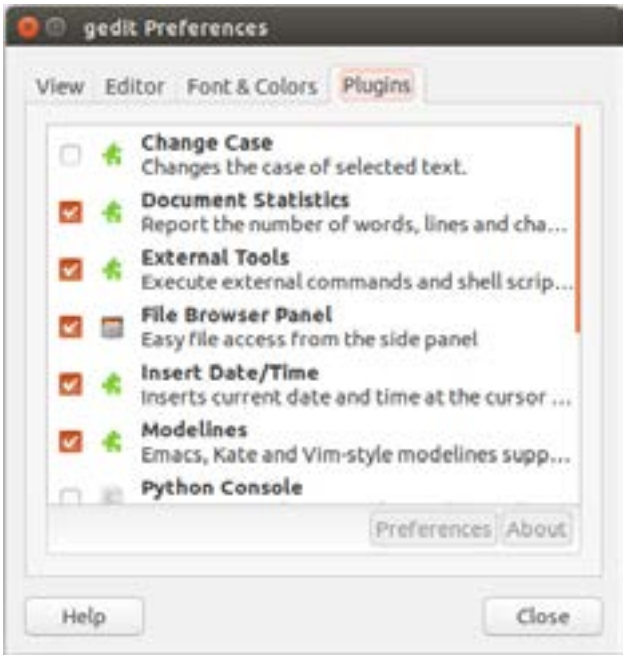
Plugins that exist on disk are “installed” but not necessarily enabled. To enable a plugin from bash you can use the following command, that modifies the preferences stored in dconf on your gnome machine:

```
gsettings set org.gnome.gedit.plugins active-plugins "['docinfo', 'time',
'zeitgeistplugin', 'spell', 'modelines', 'filebrowser']"
```

You specify the plugin names you wish to be enabled inside the list you set for the active-plugins key. The above list is of the default configuration - 6 plugins enabled.

What we noticed is that if two plugins with the same names exist in both default and local locations, the local one overrides the default one. This means that we can copy one of the default plugins’ files to the local plugins directory, modify their code to perform our desired actions when gedit loads, and we gain the runtime we looked for, patiently waiting for the user to invoke gedit with sudo. Using this overriding approach, there is no need to explicitly enable the custom plugin, because the default one is already enabled by its name, and they both share the name.

Another more crude approach, is to install a new local plugin from scratch, making it noticeable in the preferences menu:



This will also require to explicitly enable this plugin, which can be done using gnome’s gsettings utility.

Plugin Code

Gedit plugins can come in several languages, e.g. Python, C++. When using Python, you'll see a declaration in the ".plugin" file stating the loader to use:

```
Loader=python3
```

For our code to run on gedit init, we want to contain our code inside the __init__() function of the plugin, that's located in the __init__.py file inside the plugin container directory.
One could also compile its own version of an ".so" plugin and place it in the local plugins directory.

pico/nano

Pico and it's clone - nano - offer a very limited extensibility ground - only for passive objectives such as syntax highlightings. Due to this limiting approach, it remains safe from this type of attack. Maybe consider using pico/nano for editing files as root?

MITIGATION

End User

We suggest implementing the following measures on your endpoints:

OSSEC Monitoring Rules

1. Monitor modifications to the key files and folders presented in this article. Track changes and review them. For example, you can add the following rules to OSSEC syscheck configuration:

/var/ossec/etc/ossec.conf:

```
<syscheck>
      <directories check_all="yes" realtime="yes" report_changes="yes">
      ~/.config/sublime-text-3/Packages/User
      </directories>
      <directories check_all="yes" realtime="yes" report_changes="yes">
            ~/.vim/plugin/
      </directories>
      <directories check_all="yes" realtime="yes" report_changes="yes">
            ~/.emacs,
            ~/_emacs,
            ~/.emacs.d/init.el,
            ~/.xemacs,
            ~/.xemacs/init.el,
            ~/Library/Preferences/Aquamacs Emacs/Preferences.el
      </directories>
      <directories check_all="yes" realtime="yes" report_changes="yes">
            ~/.local/share/gedit/plugins
      </directories>
</syscheck>
```

Only remember to restart OSSEC after modifications:

```
sudo /var/ossec/bin/ossec-control restart
```

Other Measures

2. Deny write permissions for non-elevated users, by taking root ownership on the relevant plugins folder, e.g. *~/.config/sublime-text-3/Packages/User*.
3. Use sudoedit.

Software Developers

Change folders and file permissions models to complete the separation between the two modes.
Completely prevent loading 3rd party plugins when an editor is elevated. Alternatively, provide a manual interface to approve the elevated loading of plugins.

VENDOR RESPONSE

Acknowledging the problem? It's complicated...

Vim
No apparent intention to help remedy this issue.
Click for the full thread discussing this issue at vim developers group.

Emacs
Tagged by the developers community as notabug, security, wontfix

Sublime
- Sep 27:
  Issue description sent to the sales team & a software engineer
- Sep 28:
  ACK response, stating it was forwarded to the lead developer
- Oct 17:
  Reaching out to the vendor, asking for an update - no status update

Gedit
Bug reported on Nov 28 - remains UNCONFIRMED