Project Proposal

Aaron Greenberg, Allie Duncan, Cypress Frankenfeld, Geoff Pleiss

Our goal is to better understand compilers by editing a compiler such that it installs a "backdoor" in the operating system. During our research, we chose a Python compiler over others in order to enable us to focus our time more on understanding and editing an existing compiler. (All existing C compilers needed significant editions to be run on source code from our computer).

Techniques

Research and implementation.

Minimum Deliverable

The minimum deliverable will be a compiler that installs a backdoor at some point during the compilation process. This backdoor should not be apparent to a normal user—i.e. it will produce a program with the same functionality as a normal compiler would produce. Examples of backdoors would include: changing the permissions of the compiled executable, making the compiled executable run at sub-optimal speeds, etc.

Maximum Deliverable

The maximum deliverable will be a compiler that installs a backdoor (or several) at important points during the compilation process. This backdoor(s) will be inserted when compiling some part of a *NIX operating system. It will be invisible to a normal user, but it will allow us to have root access of the computer.

Starting up

1. Research compilers at a high level (read wikipedia)
   ✓
2. find a lightweight open source compiler that is well-documented
   ✓
3. read documentation and source code to become familiar with the compiler
   ✓
4. read examples of previous compiler backdoors
   ✓
5. start editing the compiler source code.
   ✓

Research

For the research phase of our project, we will read an overview of compilers on sites like wikipedia, and look up examples of other compiler backdoors. We will look for a lightweight, well-documented compiler to use for our project.

Preliminary Test

The built-in python compiler utilizes several editable scripts to compile python code. The scripts _init__.py, symbols.py, pyassem.py, future.py, visitor.py, ast.py, syntax.py, pycodegen.py, misc.py, consts.py, and transformer.py all have the potential to be edited to install a backdoor. To start our explorations simply, we began with the main script, __init__.py. This script simply begins the process of compilation by several top-level functions that control the compilation process. Thus, since this script is called every time a script is compiled, any addition to this file will affect every compilation. We examine the script before additions:

```
import warnings

warnings.warn("The compiler package is deprecated and removed in Python 3.x.",
              DeprecationWarning, stacklevel=2)

from compiler.transformer import parse, parseFile
from compiler.visitor import walk
from compiler.pycodegen import compile, compileFile
```

And after adding a simple print statement:

```
import warnings

warnings.warn("The compiler package is deprecated and removed in Python 3.x.",
              DeprecationWarning, stacklevel=2)

#Allie insert!
print "Allie says:"

from compiler.transformer import parse, parseFile
from compiler.visitor import walk
from compiler.pycodegen import compile, compileFile
```

As expected, this code generates “Allie says:” before any output (or generates solely “Allie says:” if there is no output). We have changed the compiler for our own purposes.
This edition, however, is neither complicated nor hard to spot. One glance at __init__.py reveils the edition. Therefore, now that we have demonstrated our ability to manipulate the compiler, we will move on to more obscure manipulations.

Challenges

The main challenges we foresee are:

- Finding a sufficiently small compiler with good documentation
    - We have initially settled upon Python's compiler. While not necessarily particularly small, it is easy to work with and has good resources
- Demonstrating that, in accordance with our goals, we all have a strong understanding of the inner workings of a compiler
    - Ideally, we will use several backdoors (by having one in each main part of the compiler) to demonstrate our comprehension of basic compiler functionality

Future Work

We have become familiar with Python's compiler and have changed the source code of the compiler to do something unexpected. We have also examined exploits (see research). Our goal this week is to discuss final compiler changes (which specific backdoors we will implement) for our report and begin on the final coding process.