Secure Johnny's Applications With Ethos

Jon A. Solworth
http://www.ethos-os.org

UIC UNIVERSITY OF ILLINOIS AT CHICAGO COLLEGE OF ENGINEERING
Department of Computer Science

Johnny wants to build robust applications

Ideally, he could

- Focus on application semantics
- Secure application with ease effort by the aid of strong system semantics
- Be able to reason about his application

In reality

Johnny carries the burden of secure his own software. This is hard in that he

- Does not have the right "mind-set" to write secure code
- Is left with no guarantee by the system, which provides low levels of abstraction
- Feels difficult to reason about security. He has to worry about vulnerabilities at different layers, backward compatibility, complex administration, and etc.

Ethos overview

- An operating system designed to ease writing and configuring robust (attack resistant) applications
- Forgoes backwards compatibility to provide only best-in-class protections
- Maximizes guaranteed protections while retaining flexibility needed for modern applications
- Currently supports applications written in Go

Ethos principles

- Security protections built in
- Simplification through providing right abstractions

POSIX C/S Program

Ethos C/S Program

Principle 1: Strong mechanism

- Network communication encrypted and authenticated
- Only authorized users and hosts are seen by servers

Principle 2: Simplification

Ethos re-designed system layer abstraction

```
serviceFd ←ipc (service, remoteHost)
```

- Makes an **encrypted** connection to a service
- $serviceName$, service to connect to; $host$, remote host name
- OS resolves $host$ to IP address and public key; NULL for localhost

```
netFd, user ←import (serviceFd)
```

- Accept an incoming connection
- Returns a file descriptor and user
- Return implies **user authenticated and authorized** by OS

```
fdSend (fd, user, program)
```

- Ethos' answer to setuid; send a file descriptor to a virtual process
- $fd[]$, tuple of file descriptors; u, user; $program$, executable

```
fd ←fdReceive ()
```

- Receive a file descriptor

```
contents ←readVar (dirFd, filename) / read (streamFd)
writeVar (dirFd, filename, contents) / write (streamFd, contents)
```

- Read or write a file in its entirety or the next object from a stream

Example: Client and server application on Ethos

Client makes a request

```
1   netFd ← ipc (serviceFd, "someService",
      "test.example.com")
2   write (netFd, "request")
3   response ← read (netFd)
```

Distributor distributes connections to virtual processes

```
4   listenFd ← advertise ("someService")
5   do forever
6       netFd, user ← import (listenFd)
7       fdSend (netFd, user, "serverVP")
```

Per-user server virtual process executes server logic

```
8   do forever
9       fd ← fdReceive ()
10      request ← read (fd)
11      // Perform service-specific work.
12      write (fd, response)
```

Johnny is happy

Johnny writes less code (zero lines of network authentication code), spends agreater fraction of his time on application semantics, can better reason about his application, and need not be concerned about many classes of security holes which in Ethos are eliminated by design.