OS Security

SELinux and Ethos

Radboud University Nijmegen, The Netherlands

Winter 2014/2015

Next week's lecture

Guest lecture by Jordy Kersten and Frans Kollée (Madison Ghurka)

31C3 Recommendations

- Tobias Engel: "SS7: Locate. Track. Manipulate."
- Karsten Nohl: "Mobile self-defense"
- Daniel J. Bernstein, Tanja Lange: "ECCHacks"
- Jacob Appelbaum, Laura Poitras: "Reconstructing narratives"
- Starbug: "Ich sehe, also bin ich ... Du"

Videos of talks online at
http://media.ccc.de/browse/congress/2014/

A short recap

- Various programs to detect/prevent intrusion (IDS, IPS, AV)

A short recap

- Various programs to detect/prevent intrusion (IDS, IPS, AV)
- Different categories of malware:
  - Virus (self-reproducing, needs host program)
  - Worm (spreading routine, no host program)
  - Trojan (no spreading routine, usually comes with useful masquerading functionality)

A short recap

- Various programs to detect/prevent intrusion (IDS, IPS, AV)
- Different categories of malware:
  - Virus (self-reproducing, needs host program)
  - Worm (spreading routine, no host program)
  - Trojan (no spreading routine, usually comes with useful masquerading functionality)
- Correction: Viruses do *not* have to print their own source code!
- Self-reproducing programs and "quines" are still related

A short recap

- Various programs to detect/prevent intrusion (IDS, IPS, AV)
- Different categories of malware:
  - Virus (self-reproducing, needs host program)
  - Worm (spreading routine, no host program)
  - Trojan (no spreading routine, usually comes with useful masquerading functionality)
- Correction: Viruses do *not* have to print their own source code!
- Self-reproducing programs and "quines" are still related
- Rootkits are used to hide the results of an attack
- Particularly dangerous: kernelspace root kits

A short recap

- Various programs to detect/prevent intrusion (IDS, IPS, AV)
- Different categories of malware:
  - Virus (self-reproducing, needs host program)
  - Worm (spreading routine, no host program)
  - Trojan (no spreading routine, usually comes with useful masquerading functionality)
- Correction: Viruses do *not* have to print their own source code!
- Self-reproducing programs and "quines" are still related
- Rootkits are used to hide the results of an attack
- Particularly dangerous: kernelspace root kits
- Even worse: bootkits infect the MBR

A short recap

- Various programs to detect/prevent intrusion (IDS, IPS, AV)
- Different categories of malware:
  - Virus (self-reproducing, needs host program)
  - Worm (spreading routine, no host program)
  - Trojan (no spreading routine, usually comes with useful masquerading functionality)
- Correction: Viruses do *not* have to print their own source code!
- Self-reproducing programs and "quines" are still related
- Rootkits are used to hide the results of an attack
- Particularly dangerous: kernelspace root kits
- Even worse: bootkits infect the MBR
- Even worse: hardware/firmware malware

A short recap

- Various programs to detect/prevent intrusion (IDS, IPS, AV)
- Different categories of malware:
  - Virus (self-reproducing, needs host program)
  - Worm (spreading routine, no host program)
  - Trojan (no spreading routine, usually comes with useful masquerading functionality)
- Correction: Viruses do *not* have to print their own source code!
- Self-reproducing programs and "quines" are still related
- Rootkits are used to hide the results of an attack
- Particularly dangerous: kernelspace root kits
- Even worse: bootkits infect the MBR
- Even worse: hardware/firmware malware
- Malware detection either signature-based or heuristic
- Code polymorphism (and self-modifying code) to counter signature-based AV

A short recap

- Various programs to detect/prevent intrusion (IDS, IPS, AV)
- Different categories of malware:
  - Virus (self-reproducing, needs host program)
  - Worm (spreading routine, no host program)
  - Trojan (no spreading routine, usually comes with useful masquerading functionality)
- Correction: Viruses do *not* have to print their own source code!
- Self-reproducing programs and "quines" are still related
- Rootkits are used to hide the results of an attack
- Particularly dangerous: kernelspace root kits
- Even worse: bootkits infect the MBR
- Even worse: hardware/firmware malware
- Malware detection either signature-based or heuristic
- Code polymorphism (and self-modifying code) to counter signature-based AV
- AV can hurt security: larger attack surface, false positives, user perception of security

LSM

- Linux security traditionally follows the UNIX security model
- Around 2000, various projects worked on MAC (and generally stronger security) for Linux
- Linus Torvalds about inclusion of SELinux: “make it a module”

LSM

- Linux security traditionally follows the UNIX security model
- Around 2000, various projects worked on MAC (and generally stronger security) for Linux
- Linus Torvalds about inclusion of SELinux: “make it a module”
- Since Kernel 2.6: API for *Linux Security Modules* (LSMs)
- Hooks to module functions when accessing security-critical resources

LSM

- Linux security traditionally follows the UNIX security model
- Around 2000, various projects worked on MAC (and generally stronger security) for Linux
- Linus Torvalds about inclusion of SELinux: “make it a module”
- Since Kernel 2.6: API for *Linux Security Modules* (LSMs)
- Hooks to module functions when accessing security-critical resources
- An LSM sets function pointers in a data structure called `security_operations`
- Global table of this type called `security_ops` defined in `include/linux/security.h`

Criticism of LSM

LSM is in the mainline kernel and various LSM implemetations exist, however, there is some criticism of the API:

- Small overhead even if no LSM is loaded

Criticism of LSM

LSM is in the mainline kernel and various LSM implemetations exist, however, there is some criticism of the API:

- Small overhead even if no LSM is loaded
- LSM is designed for access control, but can be abused, for example, for bypassing GPL license

Criticism of LSM

LSM is in the mainline kernel and various LSM implemetations exist, however, there is some criticism of the API:

- Small overhead even if no LSM is loaded
- LSM is designed for access control, but can be abused, for example, for bypassing GPL license
- "Because LSM is compiled and enabled in the kernel, its symbols are exported. Thus, every rootkit and backdoor writer will have every hook he ever wanted in the kernel." (https://grsecurity.net/lsm.php)

Criticism of LSM

LSM is in the mainline kernel and various LSM implemetations exist, however, there is some criticism of the API:

- Small overhead even if no LSM is loaded
- LSM is designed for access control, but can be abused, for example, for bypassing GPL license
- "Because LSM is compiled and enabled in the kernel, its symbols are exported. Thus, every rootkit and backdoor writer will have every hook he ever wanted in the kernel." (https://grsecurity.net/lsm.php)
- LSM provides hooks only for access control
- Systems like grsecurity and RSBAC need more than just access control

Criticism of LSM

LSM is in the mainline kernel and various LSM implemetations exist, however, there is some criticism of the API:

- Small overhead even if no LSM is loaded
- LSM is designed for access control, but can be abused, for example, for bypassing GPL license
- "Because LSM is compiled and enabled in the kernel, its symbols are exported. Thus, every rootkit and backdoor writer will have every hook he ever wanted in the kernel." (https://grsecurity.net/lsm.php)
- LSM provides hooks only for access control
- Systems like grsecurity and RSBAC need more than just access control
- "Stacking" multiple security modules is problematic

Criticism of LSM

LSM is in the mainline kernel and various LSM implemetations exist, however, there is some criticism of the API:

- Small overhead even if no LSM is loaded
- LSM is designed for access control, but can be abused, for example, for bypassing GPL license
- "Because LSM is compiled and enabled in the kernel, its symbols are exported. Thus, every rootkit and backdoor writer will have every hook he ever wanted in the kernel." (https://grsecurity.net/lsm.php)
- LSM provides hooks only for access control
- Systems like grsecurity and RSBAC need more than just access control
- "Stacking" multiple security modules is problematic
- LSM hooks expose kernel internal data structures as parameters

Implementations of LSM

- AppArmor (see lecture on virtualization)
- Linux Intrusion Detection System (LIDS)
- POSIX capabilitites
- Simplified Mandatory Access Control Kernel (Smack)
- TOMOYO
- Security-Enhanced Linux (SELinux)

Ethos OS

- All previous security features of an OS were "add-on"
- System calls, shells interface, utilities etc. implement the POSIX standards for UNIX OSs
- UNIX goes back to the 70s, not designed for security

Ethos OS

- All previous security features of an OS were "add-on"
- System calls, shells interface, utilities etc. implement the POSIX standards for UNIX OSs
- UNIX goes back to the 70s, not designed for security
- Ethos is a new operating-system design
- Project started in 2007 by Jon Solworth at UIC
- Ethos does *not* implement the POSIX standard, it's radically "clean-slate"
- Ethos is designed for security

Motivation

- "A secure OS by itself is meaningless"
- Main goal and motivation of Ethos: make it easy to write *robust applications*:

  A program is robust if it continues to operate as intended even in the face of an intelligent adversary.

Motivation

- "A secure OS by itself is meaningless"
- Main goal and motivation of Ethos: make it easy to write *robust applications*:

  A program is robust if it continues to operate as intended even in the face of an intelligent adversary.
- Typical security-critical application-level failures:
  - Fail to provide adequate security services, e.g., encryption, authentication, authorization

Motivation

- "A secure OS by itself is meaningless"
- Main goal and motivation of Ethos: make it easy to write *robust applications*:

  A program is robust if it continues to operate as intended even in the face of an intelligent adversary.
- Typical security-critical application-level failures:
  - Fail to provide adequate security services, e.g., encryption, authentication, authorization
  - Programming flaws like buffer overflows, race conditions, missing or incorrectly used security services

Motivation

- "A secure OS by itself is meaningless"
- Main goal and motivation of Ethos: make it easy to write *robust applications*:

  A program is robust if it continues to operate as intended even in the face of an intelligent adversary.
- Typical security-critical application-level failures:
  - Fail to provide adequate security services, e.g., encryption, authentication, authorization
  - Programming flaws like buffer overflows, race conditions, missing or incorrectly used security services
  - Failures at the intersection of mechanisms

Motivation

- "A secure OS by itself is meaningless"
- Main goal and motivation of Ethos: make it easy to write *robust applications*:

  A program is robust if it continues to operate as intended even in the face of an intelligent adversary.
- Typical security-critical application-level failures:
  - Fail to provide adequate security services, e.g., encryption, authentication, authorization
  - Programming flaws like buffer overflows, race conditions, missing or incorrectly used security services
  - Failures at the intersection of mechanisms
- Problem: Too much responsibility for application programmers
- Example: Hundreds of LoC to use OpenSSL in typical server applications

Motivation

- "A secure OS by itself is meaningless"
- Main goal and motivation of Ethos: make it easy to write *robust applications*:

  A program is robust if it continues to operate as intended even in the face of an intelligent adversary.
- Typical security-critical application-level failures:
  - Fail to provide adequate security services, e.g., encryption, authentication, authorization
  - Programming flaws like buffer overflows, race conditions, missing or incorrectly used security services
  - Failures at the intersection of mechanisms
- Problem: Too much responsibility for application programmers
- Example: Hundreds of LoC to use OpenSSL in typical server applications
- Solution in Ethos: provide higher-level API (system calls) that takes care of security issues
- Ethos is designed for network (Internet) applications

Design on top of Xen

- Ethos is not running on bare hardware
- Ethos is running inside the Xen Virtual Machine Monitor (VMM)
- Xen Dom0 OS is typically Linux
- Virtualization allows to run Ethos alongside Linux

Design on top of Xen

- Ethos is not running on bare hardware
- Ethos is running inside the Xen Virtual Machine Monitor (VMM)
- Xen Dom0 OS is typically Linux
- Virtualization allows to run Ethos alongside Linux
- Ethos started with Mini-OS (provided by Xen)

Design on top of Xen

- Ethos is not running on bare hardware
- Ethos is running inside the Xen Virtual Machine Monitor (VMM)
- Xen Dom0 OS is typically Linux
- Virtualization allows to run Ethos alongside Linux
- Ethos started with Mini-OS (provided by Xen)
- Additions of Ethos to Mini-OS:
  - Processes and system calls
  - Networking and Inter-process communication (IPC)
  - Filesystem
  - Cryptography
  - Authentication
  - Types
  - User-space Debugger

Design on top of Xen

- Ethos is not running on bare hardware
- Ethos is running inside the Xen Virtual Machine Monitor (VMM)
- Xen Dom0 OS is typically Linux
- Virtualization allows to run Ethos alongside Linux
- Ethos started with Mini-OS (provided by Xen)
- Additions of Ethos to Mini-OS:
  - Processes and system calls
  - Networking and Inter-process communication (IPC)
  - Filesystem
  - Cryptography
  - Authentication
  - Types
  - User-space Debugger
- Also cleaned up lots of code

“Laziness”

Building on top of Xen makes development of a new OS feasible:

- Use a Linux program called `shadowdæmon` that provides services to Ethos running in another Xen domain
- Use RPC over Xen's virtual network interfaces
- Eventually replace `shadowdæmon` by native Ethos implementations

“Laziness”

Building on top of Xen makes development of a new OS feasible:

- Use a Linux program called `shadowdæmon` that provides services to Ethos running in another Xen domain
- Use RPC over Xen's virtual network interfaces
- Eventually replace `shadowdæmon` by native Ethos implementations
- **Filesystem:** Use existing filesystem in Dom0 and `shadowdæmon` calls to read/write. `ext4` has >25000 LoC; Ethos file-system component has 1754 + 814 in `shadowdæmon`

"Laziness"

Building on top of Xen makes development of a new OS feasible:

- Use a Linux program called `shadowdæmon` that provides services to Ethos running in another Xen domain
- Use RPC over Xen's virtual network interfaces
- Eventually replace `shadowdæmon` by native Ethos implementations
- **Filesystem:** Use existing filesystem in Dom0 and `shadowdæmon` calls to read/write. `ext4` has >25000 LoC; Ethos file-system component has 1754 + 814 in `shadowdæmon`
- **Networking:** Use ARP implementation in Dom0 with static ARP tables

"Laziness"

Building on top of Xen makes development of a new OS feasible:

- Use a Linux program called `shadowdæmon` that provides services to Ethos running in another Xen domain
- Use RPC over Xen's virtual network interfaces
- Eventually replace `shadowdæmon` by native Ethos implementations
- **Filesystem:** Use existing filesystem in Dom0 and `shadowdæmon` calls to read/write. `ext4` has >25000 LoC; Ethos file-system component has 1754 + 814 in `shadowdæmon`
- **Networking:** Use ARP implementation in Dom0 with static ARP tables
- **Drivers:** >5 Mio. LoC for drivers in Linux. Ethos' network driver is 462 LoC, console driver is 296 LoC

"Laziness"

Building on top of Xen makes development of a new OS feasible:

- Use a Linux program called `shadowdæmon` that provides services to Ethos running in another Xen domain
- Use RPC over Xen's virtual network interfaces
- Eventually replace `shadowdæmon` by native Ethos implementations
- **Filesystem:** Use existing filesystem in Dom0 and `shadowdæmon` calls to read/write. `ext4` has >25000 LoC; Ethos file-system component has 1754 + 814 in `shadowdæmon`
- **Networking:** Use ARP implementation in Dom0 with static ARP tables
- **Drivers:** >5 Mio. LoC for drivers in Linux. Ethos' network driver is 462 LoC, console driver is 296 LoC
- **Debugging:** Use `gdbsx` debugger of Xen

"Laziness"

Building on top of Xen makes development of a new OS feasible:

- Use a Linux program called `shadowdæmon` that provides services to Ethos running in another Xen domain
- Use RPC over Xen's virtual network interfaces
- Eventually replace `shadowdæmon` by native Ethos implementations
- **Filesystem:** Use existing filesystem in Dom0 and `shadowdæmon` calls to read/write. `ext4` has >25000 LoC; Ethos file-system component has 1754 + 814 in `shadowdæmon`
- **Networking:** Use ARP implementation in Dom0 with static ARP tables
- **Drivers:** >5 Mio. LoC for drivers in Linux. Ethos' network driver is 462 LoC, console driver is 296 LoC
- **Debugging:** Use `gdbsx` debugger of Xen
- **Testing:** "Fast" to get a prototype working, can automate testing from Dom0

Pitfalls of using a VMM

- VMM itself can have bugs (Ethos helped one such problem)
- Dom0 in Xen has direct access to
  1. hardware I/O devices
  2. the virtual memory of other virtual machines

Pitfalls of using a VMM

- VMM itself can have bugs (Ethos helped one such problem)
- Dom0 in Xen has direct access to
  1. hardware I/O devices
  2. the virtual memory of other virtual machines
- Address problem 1 by encrypting all data sent to communication devices and file systems
- Problem 2 could be addressed in Xen by encrypting memory pages before Dom0 access

Pitfalls of using a VMM

- VMM itself can have bugs (Ethos helped one such problem)
- Dom0 in Xen has direct access to
  1. hardware I/O devices
  2. the virtual memory of other virtual machines
- Address problem 1 by encrypting all data sent to communication devices and file systems
- Problem 2 could be addressed in Xen by encrypting memory pages before Dom0 access
- Long-term plans (ideas) for Ethos:
  - Microkernel implementation
  - Develop minimalist VMM
  - Verify VMM

What Ethos ensures

Protection mechanisms are *compulsory*, most important ones:

- **P1:** Processes cannot change owners; instead, processes spawn special children that run as a different owner from inception

What Ethos ensures

Protection mechanisms are *compulsory*, most important ones:

- **P1:** Processes cannot change owners; instead, processes spawn special children that run as a different owner from inception
- **P2:** Applications do not have access to secret keys; instead, Ethos isolates keys and provides access to cryptographic operations through system calls

What Ethos ensures

Protection mechanisms are *compulsory*, most important ones:

- **P1:** Processes cannot change owners; instead, processes spawn special children that run as a different owner from inception
- **P2:** Applications do not have access to secret keys; instead, Ethos isolates keys and provides access to cryptographic operations through system calls
- **P3:** All network connections are authenticated

What Ethos ensures

Protection mechanisms are *compulsory*, most important ones:

- **P1:** Processes cannot change owners; instead, processes spawn special children that run as a different owner from inception
- **P2:** Applications do not have access to secret keys; instead, Ethos isolates keys and provides access to cryptographic operations through system calls
- **P3:** All network connections are authenticated
- **P4:** Authentication uses strong techniques

What Ethos ensures

Protection mechanisms are *compulsory*, most important ones:

- **P1:** Processes cannot change owners; instead, processes spawn special children that run as a different owner from inception
- **P2:** Applications do not have access to secret keys; instead, Ethos isolates keys and provides access to cryptographic operations through system calls
- **P3:** All network connections are authenticated
- **P4:** Authentication uses strong techniques
- **P5:** Confidentiality of authentication databases is not essential to security because Ethos uses public-key cryptography

What Ethos ensures

Protection mechanisms are *compulsory*, most important ones:

- **P1:** Processes cannot change owners; instead, processes spawn special children that run as a different owner from inception
- **P2:** Applications do not have access to secret keys; instead, Ethos isolates keys and provides access to cryptographic operations through system calls
- **P3:** All network connections are authenticated
- **P4:** Authentication uses strong techniques
- **P5:** Confidentiality of authentication databases is not essential to security because Ethos uses public-key cryptography
- **P6:** All communication made (client-side/local user) or received (server-side/remote user) are subject to authorization based on the requesting host and user

What Ethos ensures

Protection mechanisms are *compulsory*, most important ones:

- **P1:** Processes cannot change owners; instead, processes spawn special children that run as a different owner from inception
- **P2:** Applications do not have access to secret keys; instead, Ethos isolates keys and provides access to cryptographic operations through system calls
- **P3:** All network connections are authenticated
- **P4:** Authentication uses strong techniques
- **P5:** Confidentiality of authentication databases is not essential to security because Ethos uses public-key cryptography
- **P6:** All communication made (client-side/local user) or received (server-side/remote user) are subject to authorization based on the requesting host and user
- **P7:** All data written to disk or network devices is protected using strong cryptography

Etypes

- Typical input to programs in UNIX are byte arrays (from the network, from files, from stdin)
- Parsing to typed inputs is left to applications
- Improper handling of ill-formed (e.g., malicious) inputs is common source of security issues

Etypes

- Typical input to programs in UNIX are byte arrays (from the network, from files, from stdin)
- Parsing to typed inputs is left to applications
- Improper handling of ill-formed (e.g., malicious) inputs is common source of security issues
- Ethos offers *distributed types* in the *Etypes* subsystem:
  - A notation, ETN, for specifying types
  - a machine-readable type description ("type graph")
  - A single wire format (ETE)
  - Tools (userspace and kernelspace) to transform ETN into code that will encode, decode, and recognize types
  - Extensions to `read` and `write` system calls to check input and output

Etypes

- Typical input to programs in UNIX are byte arrays (from the network, from files, from stdin)
- Parsing to typed inputs is left to applications
- Improper handling of ill-formed (e.g., malicious) inputs is common source of security issues
- Ethos offers *distributed types* in the *Etypes* subsystem:
  - A notation, ETN, for specifying types
  - a machine-readable type description ("type graph")
  - A single wire format (ETE)
  - Tools (userspace and kernelspace) to transform ETN into code that will encode, decode, and recognize types
  - Extensions to `read` and `write` system calls to check input and output
- Programs specify what input types they allow
- Validity of input (and outputs) enforced by OS

Available types

- Primitive types (`byte`, `int32`)

Available types

- Primitive types (`byte`, `int32`)
- Vectors (tuples, strings, arrays)

Available types

- Primitive types (`byte`, `int32`)
- Vectors (tuples, strings, arrays)
- Composites (structs, dictionaries, unions)

Available types

- Primitive types (`byte`, `int32`)
- Vectors (tuples, strings, arrays)
- Composites (structs, dictionaries, unions)
- Pointers

Available types

- Primitive types (`byte`, `int32`)
- Vectors (tuples, strings, arrays)
- Composites (structs, dictionaries, unions)
- Pointers
- RPC interfaces

Available types

- Primitive types (`byte`, `int32`)
- Vectors (tuples, strings, arrays)
- Composites (structs, dictionaries, unions)
- Pointers
- RPC interfaces
- Any

Directories and types

- Directories "know" what types they may contain
- Typing is enforced for all non-directory contents of a directory
- Network connections, IPC, are using the filesystem
- Example: All file objects in a directory for IPv4 addresses must have type `int32`
- "Any" type allows traditional directories

System calls

UNIX		**Ethos**	
`mkdir`	Create directory, given path and mode	`createDirectory`	Create directory, given parent FD, name, label, and type hash
`open`	Open file for successive read/write	`read/writeVar`	Read/Write object in its entirety
`read`	Read a number of bytes	`read`	Read an object
`write`	Write a number of bytes	`write`	Write an object
`seek`	Seek within a file	`n/a`	Seek at object level by using directory as streaming object

Networking in Ethos

Server

```
fdListen = advertise("ping"); // bind
fd , user = import(fdListen); // accept
write (fd, "Hello");
```

Client

```
// connect
fd = ipc("ping", "example.com");
v = read(fd);
```

Networking in Ethos

Server

```
fdListen = advertise("ping"); // bind
fd , user = import(fdListen); // accept
write (fd, "Hello");
```

Client

```
// connect
fd = ipc("ping", "example.com");
v = read(fd);
```

- Syntax similar to POSIX, but with some cleanups (names instead of numbers, remove excess calls)

Networking in Ethos

Server

```
fdListen = advertise("ping"); // bind
fd , user = import(fdListen); // accept
write (fd, "Hello");
```

Client

```
// connect
fd = ipc("ping", "example.com");
v = read(fd);
```

- Syntax similar to POSIX, but with some cleanups (names instead of numbers, remove excess calls)
- Core difference: semantics! (e.g., `user` for `import` is the *remote* user)

Properties of Ethos networking

- All network communication encrypted and authenticated
- Uses Networking and Cryptography library (NaCl) for crypto
- MinimaLT network protocol (faster than unencrypted TCP/IP)
- Authentication is public-key based
  - user IDs are public keys
  - users can mint as many identities as they like

Properties of Ethos networking

- All network communication encrypted and authenticated
- Uses Networking and Cryptography library (NaCl) for crypto
- MinimaLT network protocol (faster than unencrypted TCP/IP)
- Authentication is public-key based
  - user IDs are public keys
  - users can mint as many identities as they like
- Services are named by paths in the file system (readability)
- Directory authorizes both
  - hosts (incoming and outgoing)
  - users (incoming)

Properties of Ethos networking

- All network communication encrypted and authenticated
- Uses Networking and Cryptography library (NaCl) for crypto
- MinimaLT network protocol (faster than unencrypted TCP/IP)
- Authentication is public-key based
  - user IDs are public keys
  - users can mint as many identities as they like
- Services are named by paths in the file system (readability)
- Directory authorizes both
  - hosts (incoming and outgoing)
  - users (incoming)
- All data passed through Ethos is directory-specified type
  - avoid input vulnerabilities
  - encoder/decoder makes automatically achieves host-independence

Properties of Ethos networking

- All network communication encrypted and authenticated
- Uses Networking and Cryptography library (NaCl) for crypto
- MinimaLT network protocol (faster than unencrypted TCP/IP)
- Authentication is public-key based
  - user IDs are public keys
  - users can mint as many identities as they like
- Services are named by paths in the file system (readability)
- Directory authorizes both
  - hosts (incoming and outgoing)
  - users (incoming)
- All data passed through Ethos is directory-specified type
  - avoid input vulnerabilities
  - encoder/decoder makes automatically achieves host-independence
- Ethos uses a distributed efficient public-key infrastructure called sayl

Implications

- Attackers cannot read/modify network communication
- Supports anonymous or pseudonymed users
- Unwanted communication eliminated before application code

Implications

- Attackers cannot read/modify network communication
- Supports anonymous or pseudonymed users
- Unwanted communication eliminated before application code
- Zero LoC in applications for crypto and type conversions
- Applications cannot bypass security services

Implications

- Attackers cannot read/modify network communication
- Supports anonymous or pseudonymed users
- Unwanted communication eliminated before application code
- Zero LoC in applications for crypto and type conversions
- Applications cannot bypass security services
- Semantics eliminate many security holes
- Simplicity from deep integration of authentication, authorization, and networking

Present and future work in Ethos

Present

- Nearly complete prototype
- Ported Go programming language to Ethos
- Beginning of user-space foundation (EI shell language, graphics)
- Some small applications
- Close to releasing MinimaLT and sayl

Present and future work in Ethos

Present

- Nearly complete prototype
- Ported Go programming language to Ethos
- Beginning of user-space foundation (EI shell language, graphics)
- Some small applications
- Close to releasing MinimaLT and sayl

Future

- From prototype to production kernel
- Develop EI, tools, graphics
- Build secure Ethos applications

Advertisement

Interested in working on Ethos?

Jon is looking for students who are interested in working on Ethos in their

- Bachelor's thesis
- Master's thesis
- Ph.D. thesis

More details on Ethos are on

http://ethos-os.org