Hello!

LINUS HENZE
Key Steal

www.pinauten.de

Objective by the Sea
ABOUT ME

• Linus Henze (@LinusHenze)
• Independent iOS and macOS security researcher from Germany
• CS student at Universität Koblenz
• Website: pinauten.de
• Exploits can be found on GitHub: github.com/LinusHenze
AGENDA

• Let’s talk about the Keychain
• Keychain Internals
• Exploiting the Keychain
• Apple's fix
• Demonstration
LET’S TALK ABOUT THE KEYCHAIN

HIGH LEVEL VIEW ON THE KEYCHAIN
WHAT IS THE KEYCHAIN?

• Central place for your passwords/certificates/...
• One Keychain per user + System Keychain
• Additionally, each user has an iCloud Keychain
  • Not a normal Keychain: different implementation and APIs
  • Not in scope of this talk
LOGIN KEYCHAIN

- Login Keychain
  - Located in ~/Library/Keychains/login.keychain-db
- Usually encrypted using your login password
  - Automatically unlocked on login
- Used by many Apps and system services
- Contains all your personal passwords
SYSTEM KEYCHAIN

- System Keychain
  - Located in /Library/Keychains/System.keychain
  - Encrypted using a per-device key
    - Key stored in /var/db/SystemKey, can only be read by root
- Mainly stores WiFi passwords and certificates
- Only accessible by administrators
ADVANTAGES/DISADVANTAGES

• Simple (and safe) way to store credentials
• Safe way to share credentials with other Apps
• Only need to remember the login password

• Single point of failure
• Large attack surface
  • Process responsible for the Keychain is doing a lot of things
• Metadata (e.g. usernames) stored unencrypted, only passwords/keys/secure notes are encrypted
Accessing the Keychain

High Level API
KEYCHAIN ITEMS

- Every entry in the Keychain is a Keychain Item
- Keychain Items have an associated "class"
  - Internet Password, Generic Password
  - Key (public/private)
  - Certificate, Identity (Certificate + private Key)
- Depending on their class, Keychain Items may have multiple attributes
  - e.g. the Username and Server for Internet Passwords or a Type (used for Secure Notes, which are Generic Passwords)
USEFUL APIS

- `SecItemCopyMatching`: Allows you to search the keychain for items having certain attributes (e.g. class, username, server etc.)
- `SecItemAdd`: Create a new item with attributes
- `SecItemDelete`: Delete an item
- `SecItemUpdate`: Search for items and update them
import Foundation
import Security

/*
 * Setup our query
 * We want to get every Internet Password Item (without requesting the actual password as the user would need to allow that)
 * *
 * Class: Internet Password
 * Limit: None (return all Items that are of the Internet Password class)
 * Return Attributes: True so that we get the Account Names
 * Return Data: False because that would show a Keychain Prompt
 */
let query: [CFString: Any] = [kSecClass: kSecClassInternetPassword,
                        kSecMatchLimit: kSecMatchLimitAll,
                        kSecReturnAttributes: true,
                        kSecReturnData: false]

var items: CFTyperef!
let status = SecItemCopyMatching(query as CFDictionary, &items) // Query the Keychain

guard status == errSecSuccess else {
    /* Proper error handling goes here... */
    fatalError("Failed to get Keychain Items")
}

print("Found the following Internet Accounts in your keychain:")

for item in items as! [[String: Any]] {
    let username = item["acct"] as? String ?? "<No username>"
    let server = item["srvr"] as? String ?? "<No server>"
    print("Username: \(username) - Server: \(server)")
}
/*
* Setup our query
* We want to get every Internet Password Item (without requesting the actual password
* as the
* user would need to allow that)
* 
* Class: Internet Password
* Limit: None (return all Items that are of the Internet Password class)
* Return Attributes: True so that we get the Account Names
* Return Data: False because that would show a Keychain Prompt
*/

let query: [CFString: Any] = [kSecClass: kSecClassInternetPassword,
                             kSecMatchLimit: kSecMatchLimitAll,
                             kSecReturnAttributes: true,
                             kSecReturnData: false]

var items: CFTypeRef!
let status = SecItemCopyMatching(query as CFDictionary, &items) // Query the Keychain
import Foundation
import Security

/*
 * Setup our query
 * We want to get every Internet Password Item (without requesting the actual password as the user would need to allow that)
 * 
 * Class: Internet Password
 * Limit: None (return all Items that are of the Internet Password class)
 * Return Attributes: True so that we get the Account Names
 * Return Data: False because that would show a Keychain Prompt
 */
let query: [CFString: Any] = [kSecClass: kSecClassInternetPassword, 
 kSecMatchLimit: kSecMatchLimitAll, 
 kSecReturnAttributes: true, 
 kSecReturnData: false]

var items: CFTypeRef!
let status = SecItemCopyMatching(query as CFDictionary, &items) // Query the Keychain

guard status == errSecSuccess else {
    /* Proper error handling goes here... */
    fatalError("Failed to get Keychain Items")
}

print("Found the following Internet Accounts in your keychain:")

for item in items as! [[String: Any]] {
    let username = item["acct"] as? String ?? "<No username>"
    let server = item["srvr"] as? String ?? "<No server>"
    print("Username: 
 (username) - Server: 
 (server)"")
}
KEYCHAIN INTERNALS

HOW IT WORKS
KEYCHAIN INTERNALS

- `securityd` is the daemon responsible for the keychain
  - Holds encryption keys for all unlocked keychains
  - Performs access control
- Security Framework (implementing the high level keychain APIs) communicates with `securityd` through low level MIG APIs
  - MIG (Mach Interface Generator): Implements RPC over mach messages
Communication between Apps and securityd
MACH PORTS

- Message queues, implemented by the kernel
- Works like a mailbox
  - Many senders, each holding a send right
  - Exactly one receiver, holding the receive right
    - Receiver has to tell other processes where they should send their messages to
- Referenced by mach port "names"
  - Integers, at least in userspace
  - Unique for very process
MACH TASK PORTS

- Special type of mach port
- Every process has one
- Kernel listens on these ports
- Allows to modify the process
  - i.e. map/unmap/modify memory and other stuff
- Can be used to identify a process
- Automatically deleted once the process dies
MACH MESSAGES

• Structured data sent to a mach port
  • Header: Basic information like where to send the message to, size, message ID and an optional reply port
  • Body: May contain send/receive rights for mach ports and arbitrary data
• Queued in the kernel until retrieved by the receiver
  • Unless there are already too many messages in the queue...
simplified, in reality it's not that easy...

SECURITYD EXAMPLE

<table>
<thead>
<tr>
<th>Query Keychain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query:</td>
</tr>
<tr>
<td>Password for Twitter, Username: LinusHenze</td>
</tr>
<tr>
<td>Reply to Port:</td>
</tr>
<tr>
<td>0x4142</td>
</tr>
<tr>
<td>(Twitter's mach port as seen by securityd)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query Keychain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query:</td>
</tr>
<tr>
<td>Password for Twitter, Username: LinusHenze</td>
</tr>
<tr>
<td>Reply to Port:</td>
</tr>
<tr>
<td>0x1337</td>
</tr>
<tr>
<td>(Twitter's mach port as seen by Twitter)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Twitter</th>
<th>securityd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Result</td>
</tr>
<tr>
<td>Password: 123456</td>
<td>Password: 123456</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Twitter</th>
<th>securityd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter's mach port</td>
<td>securityd's mach port</td>
</tr>
</tbody>
</table>
SECURITYD - MACH PORT

• How do we get a send right to securityd's mach port?
SECURITYD - MACH PORT

How do we get a send right to securityd's mach port?

- Through launchd!
LAUNCHD

• Init process on macOS
• Every process inherits a "bootstrap port" from its parent
  • This is almost always launchd's mach port
• All services register with launchd
  • Just need to ask launchd to give us a send right to securityd's mach port
LAUNCHD EXAMPLE

LAUNCHD EXAMPLE

Result
Port:
0xABCD
(securityd’s mach port as seen by launchd)

Launchd

Lookup service
Service:
com.apple.SecurityServer
Reply to Port:
0x1234
(Twitter’s mach port as seen by launchd)

Twitter's mach port

Result
Port:
0xC0DE
(securityd’s mach port as seen by launchd)

Launchd's mach port

Twitter

Lookup service
Service:
com.apple.SecurityServer
Reply to Port:
0x1337
(Twitter’s mach port as seen by Twitter)
simplified, in reality it's not that easy...
Keychain Access Control
KEYCHAIN ACCESS CONTROL

• Each Keychain Item has an ACL (Access Control List)
  • List of applications that may access the item without a password prompt
• Can only be changed by the user or Apps already in the item's ACL
• But how is it enforced?
KEYCHAIN ACCESS CONTROL

• Each Keychain Item has an ACL (Access Control List)
  • List of applications that may access the item without a password prompt
• Can only be changed by the user or Apps already in the item's ACL
• But how is it enforced?
  • By requiring Apps to submit their task port before being allowed to do anything else
SECURITYD EXAMPLE

Create Session

Task Port:
0xABCD
(Twitter’s task port as seen by securityd)

Reply to Port:
0x1337
(Twitter’s mach port as seen by securityd)

Result
OK

securityd

Create Session

Task Port:
0xC0DE
(Twitter’s task port as seen by Twitter)

Reply to Port:
0x4142
(Twitter’s mach port as seen by securityd)

Result
OK

Twitter

Twitter’s mach port

securityd’s mach port
simplified, in reality it's not that easy...
TIME TO EXPLOIT THE KEYCHAIN

KEYSTEAL VS KEYCHAIN
HOW I FOUND THE BUG

• Needed a sandbox escape (so I can do something cool when I find my next WebKit vulnerability)
• Looked into WebContent's sandbox profile
  • Is allowed to access the "com.apple.SecurityServer" service (securityd)
    • I just had to look into this (because of the name)
• It's not what I hoped for, but without this bug I wouldn't be here ;)

WHERE IS THE BUG
Article

Hosting Guest Code

Securely launch and manage plug-ins and other executable entities, known as guest code, from within your app acting as a host.

Overview

The functions in this section are called only by code that is hosting guests. In the context of code signing, a host is code that creates, launches, and manages other code—its guests. A host must do this without compromising its own integrity. As part of that duty, it maintains state for each of its guests and answers questions about them.
HOSTING GUEST CODE

- Never heard of this feature?
  - Me neither!
- Implemented in securityd
- Apparently, you should be able to use it to host guest code and tell the system about it
  - But it's completely broken...
  - And also has a nice vulnerability
Function

SecHostSetHostingPort

Tells code signing services that the calling code will directly respond to hosting inquiries over the given port.

Declaration

OSStatus SecHostSetHostingPort(mach_port_t hostingPort, SecCSFlags flags);

Parameters

hostingPort
/\  
/\ Register a hosting API service port where the host will dynamically 
/\ answer hosting queries from interested parties. This switches the process 
/\ to dynamic hosting mode, and is incompatible with proxy hosting. 
/\ void CodeSigningHost::registerCodeSigning(mach_port_t hostingPort, SecCSFlags flags)
{ 
    StLock<Mutex> _(mLock); 
    switch (mHostingState) {
    case noHosting:
        mHostingPort = hostingPort;
        mHostingState = dynamicHosting;
        secnotice("SecServer", "%d host register: %d", mHostingPort.port(), mHostingPort.port());
        break;
    default:
        MacOSError::throwMe(errSecCSHostProtocolContradiction);
    }
THE BUG

// Reset Code Signing Hosting state.
// This turns hosting off and clears all children.

void CodeSigningHost::reset()
{
    StLock<Mutex> _(_lock);
    switch (mHostingState) {
    case noHosting:
        break; // nothing to do
    case dynamicHosting:
        mHostingPort.destroy();
        mHostingPort = MACH_PORT_NULL;
        secnotice("SecServer", "%d host unregister", mHostingPort.port());
        break;
    case proxyHosting:
        Server::active().remove(*this); // unhook service handler
        mHostingPort.destroy(); // destroy receive right
        mHostingState = noHosting;
        mHostingPort = MACH_PORT_NULL;
        mGuests.erase(mGuests.begin(), mGuests.end());
        secnotice("SecServer", "%d host unregister", mHostingPort.port());
        break;
    }
}
/ Reset Code Signing Hosting state.
// This turns hosting off and clears all children.

void CodeSigningHost::reset()
{
    StLock<Mutex> _(mLock);
    switch (mHostingState) {
    case noHosting:
        break; // nothing to do
    case dynamicHosting:
        mHostingPort.destroy();
        mHostingPort = MACH_PORT_NULL;
        secnotice("SecServer", "%d host unregister", mHostingPort.port());
        break;
    case proxyHosting:
        Server::active().remove(*this); // unhook service handler
        mHostingPort.destroy(); // destroy receive right
        mHostingState = noHosting;
        mHostingPort = MACH_PORT_NULL;
        mGuests.erase(mGuests.begin(), mGuests.end());
        secnotice("SecServer", "%d host unregister", mHostingPort.port());
        break;
    }
}
THE BUG

• We can give securityd a send right to an arbitrary port
• When our session is destroyed, mach_port_destroy is called on the port
  • Should have been mach_port_deallocate ...
• Causes ALL references to the port being destroyed instead of just one
  • Can be used to free an arbitrary port in securityd
  • And replace it afterwards...
ATTACK PLAN

• Free an arbitrary port in securityd
• ???
SECURITYD SESSIONS

• As I've already said, before being able to talk to securityd, we need to create a session
• Session is tied to the task port of your process
  • Free the task port -> Interesting stuff happens
ATTACK PLAN

• Free task port of a process in securityd
• Force session to have a dangling task port
• ???
/**
// Set up a new Connection. This establishes the environment (process et al) as needed
// and registers a properly initialized Connection object to run with.
// Type indicates how "deep" we need to initialize (new session, process, or connection).
// Everything at and below that level is constructed. This is straightforward except
// in the case of session re-initialization (see below).
/**

```cpp
void Server::setupConnection(ConnectLevel type, Port replyPort, Port taskPort,
        const audit_token_t &auditToken, const ClientSetupInfo *info)
{
    Security::CommonCriteria::AuditToken audit(auditToken);

    // first, make or find the process based on task port
    RefPointer<Process> &proc = mProcesses[taskPort];
    if (proc && type == connectNewProcess) {
        // the client has amnesia - reset it
        proc->reset(taskPort, info, audit);
        proc->changeSession(audit.sessionId());
    }
    if (!proc) {
        if (type == connectNewThread) // client error (or attack)
            CssmError::throwMe(CSSM_ERRCODE_INTERNAL_ERROR);
        proc = new Process(taskPort, info, audit);
        notifyIfDead(taskPort);
        mPids[proc->pid()] = proc;
    }

    // now, establish a connection and register it in the server
    Connection *connection = new Connection(*proc, replyPort);
    if (mConnections.contains(replyPort)) // malicious re-entry attempt?
        CssmError::throwMe(CSSM_ERRCODE_INTERNAL_ERROR); //@ error code? (client error)
    mConnections[replyPort] = connection;
    notifyIfDead(replyPort);
}
```
// Screen a process setup request for an existing process.
// This means the client has requested initialization even though we remember having
// talked to it in the past. This could either be an exec(2), or the client could just
// have forgotten all about its securityd client state. Or it could be an attack...

void Process::reset(TaskPort taskPort, const ClientSetupInfo *info, const CommonCriteria::AuditToken &audit)
{
    StLock<Mutex> _(*this);
    if (taskPort != mTaskPort) {
        secnotice("SecServer", "Process %p(%d) reset mismatch (tp %d-%d)",
                    this, pid(), taskPort.port(), mTaskPort.port());
        //@@@ CssmError::throwMe(CSSM_ERRCODE_VERIFICATION_FAILURE);  // liar
    }
    setup(info);
    CFCopyRef<SecCodeRef> oldCode = processCode();

    // Note: The following will reload the code signature of the process
    // including all entitlements
    // HOWEVER, IT IS USING THE SAVED PID, NOT THE ONE OF THE PROCESS ASKING FOR REINITIALIZATION
    ClientIdentification::setup(this->pid()); // re-constructs processCode()
    if (CFEqual(oldCode, processCode())) {
        secnotice("SecServer", "%p Client reset amnesia", this);
    } else {
        secnotice("SecServer", "%p Client reset full", this);
        CodeSigningHost::reset();
    }
}
ATTACK PLAN

• Free task port of a process in securityd
• Force session to have a dangling task port
• Reinitialize session, making sure it's PID is reused by a process allowed to access the Keychain without a password prompt
  • Must have the "com.apple.private.security.allow-migration" entitlement
// If we have a KeychainPromptAclSubject, we want KeychainMigrator to have access even if we don't have the "pop ui" credential. Do the code signing check first, then process this ACL as normal.

// If we have a KeychainPromptAclSubject, we want KeychainMigrator to have access even if we don't have the "pop ui" credential. Do the code signing check first, then process this ACL as normal.

bool KeychainPromptAclSubject::validates(const AclValidationContext &ctx) const {
    Process &process = Server::process();
    if (process.checkAppleSigned() && process.hasEntitlement(migrationEntitlement)) {
        Syslog::info("bypassing keychain prompt for keychain migrator");
        secnotice("kcacl", "bypassing keychain prompt for keychain migrator");
        return true;  // migrator client -> automatic win
    }

    // Also, mark down that we evaluated a prompt ACL. We want to record this for testing even if the client did not pass credentials for UI
    // (so that tests can disable prompts but still detect if one would have popped)
    promptsValidated++;

    return SimpleAclSubject::validates(ctx);
}
OS X/iOS Entitlement Database - v0.6

As compiled by Jonathan Levin, @Morpheus

Pardon the appearance during construction and focus on functionality :-)  
Now with entitlements from iOS 9.0.2 through 12 (β12 - as good as final) 
Now with entitlements from MacOS 11.4 through MacOS 14  
.. and with DDI, and autocomplete

OS Version: MacOS 10.14

Executables possessing Entitlement: 

Entitlements by Executable: 

MacOS14 Entitlement com.apple.private.security.allow-migration held by:

- AirPlayService
- CardDAVService
- CertificateService
- ExchangeService
- KeychainMigrator
- InternetAccountsMigrator
- mdmclient

Entitlement data harvested automatically by JTool --ent.  
This is a work in progress. Suggestions for improvement are welcome at the NewOSxBook.com forum

newosxbook.com/ent.jl?ent=com.apple.private.security.allow-migration&osVer=MacOS14
ATTACK PLAN

• Free task port of a process in securityd
• Force session to have a dangling task port
• Reinitialize session, making sure it's PID is reused by a process allowed to access the Keychain without a password prompt
  • Must have the "com.apple.private.security.allow-migration" entitlement
  • e.g. /System/Library/InternetAccounts/internetAccountsMigrator
• Access Keychain without password prompt!
RECLAIM SESSION

• After freeing the task port, we won't have access to our session anymore
  • Need to reclaim our session
• Can be done by sending securityd a huge number of ports, hoping one of them gets the same number as our task port had
  • Use this new fake task port to access our session
ATTACK PLAN

1. Create three processes: A, B and C
2. B should create a session with securityd
3. Send task port of B to C
4. Let C free B's task port in securityd
5. B should now reclaim it's session by sending securityd many ports, hoping one of them will get the same number as B's task port had
6. Send this fake task port to A (receive right!)
7. B should exec internetAccountsMigrator
   7.1. Reclaimed session won't be deleted as A now owns the fake task port which therefore won't be deleted
8. A can now reset B's session using the fake task port
   8.1. Causes the entitlements of internetAccounts migrator to be loaded
9. Use fake task port to access keychain!!!
ATTACK PLAN

1. Create three processes: A, B and C
2. B should create a session with securityd
NEW SESSION
ATTACK PLAN

1. Create three processes: A, B and C ✓
2. B should create a session with securityd ✓
3. Send task port of B to C
ATTACK PLAN

1. Create three processes: A, B and C ✓
2. B should create a session with securityd ✓
3. Send task port of B to C ✓
4. Let C free B's task port in securityd
ATTACK PLAN

1. Create three processes: A, B and C ✓
2. B should create a session with securityd ✓
3. Send task port of B to C ✓
4. Let C free B's task port in securityd ✓
5. B should now reclaim its session by sending securityd many ports, hoping one of them will get the same number as B's task port had
RECLAIMING - SETUP

Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Task Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
</tr>
</tbody>
</table>

Session with KeySteal Helper

Connections

Conn

KeySteal

KeySteal Helper

TP
Conn
Conn

Task Port
Connection Port
Connection Port 2
RECLAIMING
RECLAIMING
RECLAIMING
AFTER SOME TIME...
RECLAIMING
RECLAIMING
RECLAIMING
ATTACK PLAN

1. Create three processes: A, B and C ✔
2. B should create a session with securityd ✔
3. Send task port of B to C ✔
4. Let C free B's task port in securityd ✔
5. B should now reclaim its session by sending securityd many ports, hoping one of them will get the same number as B's task port had ✔
6. Send this fake task port to A (receive right!)
ATTACK PLAN

1. Create three processes: A, B and C ✓
2. B should create a session with securityd ✓
3. Send task port of B to C ✓
4. Let C free B's task port in securityd ✓
5. B should now reclaim its session by sending securityd many ports,
   hoping one of them will get the same number as B's task port had ✓
6. Send this fake task port to A (receive right!) ✓
7. B should exec internetAccountsMigrator
   7.1. Reclaimed session won't be deleted as A now owns the fake task port which therefore won't be deleted
AFTER EXEC
AFTER EXEC

Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Task Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>CN</td>
</tr>
</tbody>
</table>

Connections

PID 1337

KeySteal

I'll just call that Fake Task Port (FP) from now on

Connection/Fake TaskPort

Internet Accounts Migrator

PID 1337
ATTACK PLAN

1. Create three processes: A, B and C
2. B should create a session with securityd
3. Send task port of B to C
4. Let C free B's task port in securityd
5. B should now reclaim its session by sending securityd many ports, hoping one of them will get the same number as B's task port had
6. Send this fake task port to A (receive right!)
7. B should exec internetAccountsMigrator
8. A can now reset B's session using the fake task port
   8.1. Causes the entitlements of internetAccounts migrator to be loaded
RESET SESSION

Session
Session with KeySteal Helper
Connections

Setup Session
Task Port
Connection

securityd

KeySteal
Fake TaskPort
New Connection

Internet Accounts Migrator

PID 1337

PID 1337

Resets the Session and reloads entitlements (using PID)!
RESET SESSION
Sessions

Session S1

Task Port FP

Entitled -> No Password Required

Connections

Conn PID 1337

KeySteal

FP Conn

Internet Accounts Migrator

PID 1337

Fake TaskPort New Connection

DONE
ATTACK PLAN

1. Create three processes: A, B and C
2. B should create a session with securityd
3. Send task port of B to C
4. Let C free B's task port in securityd
5. B should now reclaim its session by sending securityd many ports, hoping one of them will get the same number as B's task port had
6. Send this fake task port to A (receive right!)
7. B should exec internetAccountsMigrator
8. A can now reset B's session using the fake task port
   8.1. Causes the entitlements of internetAccounts migrator to be loaded
9. Use fake task port to access keychain!!!
HOW DID APPLE FIX THE BUG?

KEYCHAIN IS SAFE AGAIN
WHAT APPLE SAYS

Security

Available for: macOS Sierra 10.12.6, macOS High Sierra 10.13.6, macOS Mojave 10.14.3

Impact: An application may be able to gain elevated privileges

Description: A use after free issue was addressed with improved memory management.

CVE-2019-8526: Linus Henze (pinauten.de)
APPLE'S PATCH

// Reset Code Signing Hosting state.
// This turns hosting off and clears all children.

//
void CodeSigningHost::reset()
{
    StLock< Mutex > _(mLock);
    switch (mHostingState) {
        case noHosting:
            break; // nothing to do
        case dynamicHosting:
            mHostingPort.deallocate();
            mHostingPort = MACH_PORT_NULL;
            secnotice("SecServer", "%d host unregister", mHostingPort.port());
            break;
        case proxyHosting:
            Server::active().remove(*this); // unhook service handler
            mHostingPort.destroy(); // destroy receive right
            mHostingState = noHosting;
            mHostingPort = MACH_PORT_NULL;
            mGuests.erase(mGuests.begin(), mGuests.end());
            secnotice("SecServer", "%d host unregister", mHostingPort.port());
            break;
    }
}
Screen a process setup request for an existing process. This means the client has requested intialization even though we remember having talked to it in the past. This could either be an exec(2), or the client could just have forgotten all about its securityd client state. Or it could be an attack...

```cpp
void Process::reset(TaskPort taskPort, const ClientSetupInfo *info, const CommonCriteria::AuditToken &audit) {
    StLock<Mutex> _(*this);
    if (taskPort != mTaskPort) {
        secnotice("SecServer", "%p(%d) reset mismatch (tp %d-%d)",
            this, pid(), taskPort.port(), mTaskPort.port);
        //@@@ CssmError::throwMe(CSSM_ERRCODE_VERIFICATION_FAILURE); // liar
    }
    setup(info);
    CFCopyRef<SecCodeRef> oldCode = processCode();

    // Note: The following will reload the code signature of the process
    // including all entitlements
    // Now using the generation number as well
    ClientIdentification::setup(this->pid(), this->generationNumber());
    if (CFEqual(oldCode, processCode())) {
        secnotice("SecServer", "%p Client reset amnesia", this);
    } else {
        secnotice("SecServer", "%p Client reset full", this);
        CodeSigningHost::reset();
    }
}```
KEYSTEAL ON ACTION

KEYSTEAL DEMO
Thank you!

Linus Henze
@LinusHenze
www.pinauten.de
github.com/LinusHenze/Keysteal