Automatic Configuration of Windows Security with Security Templates

Przemysław Bereziński, Tomasz Dalecki, Marek Małowidzki, Michał Mazur
Military Communication Institute, 05-130 Zegrze, Poland

Abstract: In the paper we discuss the automatic configuration of Windows operating system security. The automatic configuration mechanism is based on security templates and associated system tools. We compare a few available templates (the standard one that is included in Windows, more secure templates offered by Microsoft and a template recommended by the Polish security agency), discussing their differences and security settings. We identify areas where the templates could be improved, yielding a more secure system configuration. We also point out the shortcomings of the current template language, and propose simple yet effective syntax extensions and associated processing software. Finally, we discuss additional protection mechanisms, not covered by the security templates.

Keywords: security, Windows XP, Windows workstation, secured Windows, security templates, automatic security configuration.

1. Introduction

Both in the army, as well as the public administration offices – in all places where classified documents are processed there is a demand for a secure operating system. Microsoft Windows is probably the most commonly used environment for these purposes. It is a common belief that Windows is not secure; indeed, its default, “out-of-the-box” security configuration is insufficient. In this document, which is a part of a larger project aiming at a development of a secure workstation for a classified use, we show how Windows security, using standard mechanisms and tools, could be significantly improved. We focus on configuring the system security with security templates [1].

We would like to emphasize that the paper concerns a Windows workstation. Moreover, we assume that the computer is not a member of a Windows domain. Our experience suggests that this is a typical case for special-purpose, classified systems: They are usually standalone (not connected to a network) or, when attached to a network, additional security criteria must be fulfilled. Thus, we do not discuss the Group Policy mechanism. Additionally, some security set-
tings related to Windows domains, e.g., the [Kerberos Policy] registry section, are not taken into account.

Besides, our work focuses on Windows XP [2]. While it remains valid and useful for newer Windows versions, our work has been tested on Windows XP, as this is the most recent version accepted by the Polish security office for classified systems.

The paper is organized as follows: First, we describe the automatic security configuration mechanism. Then, we compare available security templates, we show areas not covered by these templates and we also point out shortcomings that result from the current template syntax. Next, we propose extensions – concerning both content and language syntax changes – that clearly improve templates’ capabilities. Since standard Windows tools cannot process our modified syntax, we discuss dedicated software that we have developed, able to process and apply modified templates. Finally, we propose additional protection mechanisms that supplement templates and comment on future work related to Windows security.

2. Windows Automatic System Configuration Mechanism Overview

A Windows security template is a text file, typically, using the .inf extension. As other Windows .inf files, a template contains entries (or statements) grouped into sections. An example content of a template is presented in Fig. 1.

```
[Unicode]
 Unicode=yes

[Version]
 signature="$CHICAGO$"

[System Access]
 MinimumPasswordLength=9
 PasswordComplexity=1

[Registry Values]
 MACHINE\software\microsoft\driver signing\policy=3,1

[File Security]
 "%ProgramFiles%",2,"D:PAR(A;OICI;FA;;;BA)(A;OICIIO;FA;;;CO)(A;O-ICI;FA;;;SY)(A;OICI;0x1200a9;;;BU)"
```

Figure 1. An example content of a security template (snippet)

Most sections contain entries that, together, define a security policy. A number of entry types are possible – e.g., simple assignment statements (as in the [System Access] section in the example above), registry values (the [Registry Values] section), resource descriptors defined using the Security Descriptor Definition Language (SDDL) (the [File Security] section) and group membership definitions (not shown).
The Security Descriptor Definition Language (SDDL) is employed to represent security descriptors (in other words it defines access permissions in a string format). In the example above, the SDDL string defines access rights (in this case, for a directory), which is denoted by a starting “D:” string. The following Discretionary Access Control List (DACL) flags, P and AR, specify access rights inheritance in the directory tree. The subsequent four fragments in brackets define Access Control Entries (ACEs) for four account types: built-in administrators (BA), creator-owner (CO), local system (SY) and built-in users (BU). The ACEs for BA and SY are identical and grant full rights (full access) to the resource (for the directory, its subdirectories and contained files). The ACE for CO also grants full rights, although the additional flag “IO” dictates that the ACE concerns contained items only (and does not apply to the directory, for which it is configured). The fourth account’s rights are limited to reading the directory and executing files.

Note that SDDL strings look quite complex; additionally, the same information may be represented in a few different ways (for example, flags may be replaced by their numerical values). Even in case of two exactly identical strings, it is not easy for a human to make sure that they actually are the same. This suggests that creating and editing SDDL strings in a template should be somehow supported by a dedicated tool (and, indeed, such a tool exists – see Tab. 1).

The Windows system offers the following tools for template-based security configuration: The Security Configuration Editor (SCE), which consists of two Microsoft Management Console (MMC) snap-ins (“Security Templates” and “Security Configuration and Analysis”), and a command-line tool, secedit. SCE uses a separate .inf-type file, sceregvl, for a user-friendly template description. Tab. 1 lists features supported by these tools and our tool discussed in section VI.

<table>
<thead>
<tr>
<th>Feature</th>
<th>„Security Templates”</th>
<th>„Security Config. and Analysis”</th>
<th>secedit</th>
<th>Our tool (section VI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparing a template with the system config.</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Comparing templates</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Exporting the current system configuration to a template</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Applying a template to system</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Graphical template editing (including SDDL)</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Extended template syntax support (section V)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
Template processing using the system tools is depicted in Fig. 2.

![Figure 2. Template processing using the standard Windows tools](image)

Summary: Windows offers a fairly complete set of tools for template processing. The shortcomings we have identified comprise template comparison (no support), limited ability to export the system configuration to a template (such a template would be incomplete, e.g., it would lack the [File Security] section) and the fact that functionality is divided into a few separate programs.

3. Security Templates Comparison

As a starting point, we have analyzed a few available template documents, namely:

- hisecWS.inf and rootsec.inf: The standard Windows template, which represents the highest level of security for a Windows-based workstation, combined with template that defines permissions for the root of the system drive;

- Security_WinXP_W_V12.inf: a template of the Polish security agency (version dated 18 December 2006);

- SSSLF-Desktop.inf and Standalone-SSLF-Account.inf: Templates included in Security Compliance Management Manager [3], prepared by Microsoft security experts, IT experts and American government agencies (SSLF stands for Specialized Security – Limited Functionality). (Another template, SSSLF-Laptop.inf, is a slightly different version of SSSLF-Desktop.inf for laptops.)

The (quantitative) result of comparison is presented in Tab. 2; we compare these templates in more detail in subsequent paragraphs.

The most complete (and, at the same time, most secure) template document is Security_WinXP_W_V12.inf.

It contains explicitly granted access rights to the file system and registry. It also contains the largest number of entries in most sections. For example its settings concerning user accounts, system services, event log are the most restrictive.
Even though rootsec.inf also defines permission to the file system via system root and inheritance, it does not overwrite explicit permissions that are defined for child objects.

**Tab. 2. Number of entries in sections of the analyzed template documents**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Log</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Event Audit</td>
<td>8</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>File Security</td>
<td>1</td>
<td>245</td>
<td>0</td>
</tr>
<tr>
<td>Group Membership</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Privilege Rights</td>
<td>0</td>
<td>37</td>
<td>36</td>
</tr>
<tr>
<td>Profile Description</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Registry Keys</td>
<td>0</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td>Registry Values</td>
<td>45</td>
<td>85</td>
<td>69</td>
</tr>
<tr>
<td>Security Log</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Service General Setting</td>
<td>0</td>
<td>78</td>
<td>19</td>
</tr>
<tr>
<td>Strings</td>
<td>17</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>System Access</td>
<td>11</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>System Log</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

We have carefully analyzed all additional entries from other templates that have been missing in the “leading” one. Then, we have supplemented it with the statements that we think should be included. This way, we have created a combined template that – throughout the remaining part of this paper – will be referred to as a base template.

4. **Security Templates Extensions: Statements**

In this section, we identify security settings not covered by the analyzed templates (and, thus, do not present in our base template). Using guidelines from [4] and [5], we propose additional entries that improve security in a number of areas. These areas, together with an example, are listed below.

1. **Category: TCP/IP Stack Settings (6 entries added)**
   
   **Example:** specifying that the Source Routed packets should not be forwarded:

   `[Registry Values]`  
   `MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters\DisableIPSourceRouting=4,1`
2. **Category: Local Resource Sharing via Network (14 entries added)**
   **Example:** The AutoShareServer setting specifies whether special resources called “administrative shares” may be available remotely. The value of 0 disables the administrative shares.
   
   [Registry Values]
   MACHINE\system\currentcontrolset\control\services\lanmanserver\parameters\autoshareserver=4,0

3. **Category: Built-in Windows System Firewall (1 entry added)**
   **Example:** The DoNotAllowExceptions setting allows or denies exceptions to Windows Firewall policies. The value of 1 denies any exceptions.
   
   [Registry Values]
   MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\DoNotAllowExceptions=4,1

4. **Category: Local System Security (8 entries added)**
   **Example:** changing the name of the local administrator account:
   
   [System Access]
   NewAdministratorName=“foobar”

5. **Category: Network Services (27 entries added)**
   **Example:** disabling the WINS server:
   
   [Service General Setting]
   wins,4,”...”

6. **Category: Other System Services (15 entries)**
   **Example:** disabling the Infrared Monitor service (most entries from this category disables various rarely used services):
   
   [Service General Setting]
   irmon,4,“”

**Summary:** We have supplemented our base template with 71 additional entries, which improve security and, even in the case of a separate workstation (i.e., not attached to a network), increase performance through disabling all system services that are unnecessary.

5. **Security Templates Extensions: Syntax**

   We have found reading and editing templates to be a tedious and exhausting task. First, templates contain a lot of similar entries and it is usually difficult to tell whether they are identical or not (for example, trying to spot a difference in SDDL expressions for two Windows services seems to be a futile effort). Additionally, templates are plagued with mysterious, “raw” values; “deciphering” their meaning requires a reference to a manual or another source of technical knowledge.
We have discussed a number of proposals for syntax extensions to address these problems, ranging from slight changes in the existing syntax to brand new XML-based solutions with embedded programming constructs. Finally, we have agreed that – as the template technology is commonly accepted and supported by Microsoft and a large number of professionals – the changes should be modest and consistent with the current concept. Thus, in all cases a new extension has been proposed, we have preferred simplicity and unambiguity over new capabilities. Below, we present a final list of syntax extensions together with evaluation. Due to lack of space, we present (hopefully!) self-explanatory examples instead of a formal specification of the extended notation.

1. **Extension: String Constants**
   
   **Motivation:** enabling self-commenting capabilities.
   
   **Usage:** defined in the [Strings] section; applied in registry paths, file paths and SDDL expressions.
   
   **Example:**
   ```
   [Strings]
   DEFAULT_SERVICE_RIGHTS="D:(A;;CCLCSWRPWPDTLOCRRC;;;;SY)...
   ```
   
   **Evaluation:** high. Constants not only enable defining meaningful names for strings they substitute, but also reduce verbosity – multiple instances of the same string may be replaced by a constant. Windows templates already have string constants (%NAME%) but they may be defined either in the [Strings] section or later, when the template is applied to the system, as environment variables (which introduces some ambiguity). Our constants must be defined in the document. We consider using strings in (registry) values but, as string values may contain any content, it may be difficult to avoid possible mistakes.

2. **Extension: Registry Value/Key/Path Removal**
   
   **Motivation:** removing unnecessary registry entries.
   
   **Usage:** the [Registry Values] section.
   
   **Example:** value removal allows to get rid of selected firewall exceptions. Setting DoNotAllowExceptions=4,1 only allows to disable all exceptions (the entries are preceded with a minus sign).
   ```
   [Registry Values]
   - MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List\@REGEXP(\.*)=
   - MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\GloballyOpenPorts\List\@REGEXP(\.*)=
   ```
   
   **Evaluation:** moderate. In the current template, we have found a single place where the extension may be applied (the firewall configuration). No reasonable application has been found for key/path removal.
3. **Extension: Regular Expressions (Regexps)**

**Motivation:** replacing multiple similar entries by a single regexp-based pattern.

**Usage:** service names; registry and file paths.

**Example:** a single statement, shown below, has replaced 118 entries from the base template, configuring the same access rights for all .exe programs in the System32 directory:

```
[File Security]
"%Sys32\%\REGEXP (.+\..exe)", 2, "D:PAR(A;OICI;FA;;;BA)"
(A;OICI;FA;;;SY)"
```

**Evaluation:** **high.** Using regular expressions, we have been able to greatly reduce the number of template entries. Note, however, that a template that contains regular expressions is no more context-free: It must be evaluated against the current system configuration.

4. **Extension: Embedded Functions**

**Motivation:** providing additional level of processing flexibility. Currently, the only supported function is @FOREACH(), which returns subsequent elements of a string list supplied as an argument.

**Usage:** in the same places where regular expressions are allowed.

**Example:** 78 service configuration entries from the base template have been replaced by about 30 entries in the extended template. Not only the template size has been decreased; after the changes, it is easy to notice sets of services for which security settings are identical.

```
[Service General Setting]
@FOREACH(&SERVICES_SET_1&), 4, &SERVICES_SET_1_RIGHTS&
[Strings]
SERVICES_SET_1=alg cisvc ersvc ...
SERVICES_SET_1_RIGHTS="D:(A;;CCLCSWRPWPDTLOCRRC;;SY) ..."
```

**Evaluation:** **high.** Again, @FOREACH() allows to reduce greatly the number of template entries and thus increase readability. Note, however, that defining additional functions requires special care – such a powerful mechanism may also be potentially dangerous.

**Summary:** Overall, the proposed extensions allow to reduce significantly the number of entries and, at the same time, increase readability through self-commenting capabilities of string constants and grouping identical settings in a single entry (regular expressions, the @FOREACH() function). The number of entries has been reduced from 560 in the base template to 360 in the extended one (the file size has decreased, respectively, from 50kB to 28kB). We believe that improved consistence and readability directly map to increased security.
6. Custom Template Processing Tool

We have developed dedicated software for processing our extended templates. The software has been implemented in the Microsoft .NET Framework v. 4.0 environment using the C# programming language. The package class diagram is shown in Fig. 3. (In fact, our API is interface-based but, for readability, we only show main classes.) A security template is represented by the Document class, which contains Sections and Expressions. The Parser class is responsible for reading templates from a file, while the CurrentMachine class represents an interface to the Windows system – it allows both importing the current configuration to a Document instance and applying settings contained within a Document to Windows. The Document API also provides a means to manually build or edit a template and allows comparing templates. Finally, an extended template may be evaluated: Syntax extensions are appropriately processed (e.g., regular expressions are replaced by matching entries) to get a standard Windows template (such a conversion in a reverse direction is currently not implemented).

Figure 3. The class diagram (simplified); dotted lines show relationships between classes and methods
The main concept consists in making the Document class a central API element. There are various ways of creating a template document (reading a file, reading the current system configuration, building from scratch – see Fig. 4); after a document has been created, it may be further edited, stored in a file, or applied to the system. Thus, while our software may be used as a ready template processing tool, it may also be integrated into a more complete and sophisticated application, e.g., an advanced GUI tool handling all aspects of Windows configuration.

It is also important to mention that our tool has additional capabilities with respect to the interface to Windows. Examples include file access rights analysis during the import of the current system configuration to a template and the ability to configure the whole registry (the standard system tools only operate on the HKEY_LOCAL_MACHINE branch). Thus, using the standard syntax, we are able to configure a password-protected screensaver:

```plaintext
[Registry Values]
HKEY_CURRENT_USER\Control Panel\Desktop\ScreenSaverIsSecure=1,"1"
```

![Figure 4. The template document processing](image)

**Summary:** Our application groups (and extends) the functionality offered by a number of separate, standard Windows tools; additionally, as mentioned above, the package may be used as a library for other applications. The template syntax extensions increase readability and provide new features. In this way, we propose a ready, integrated, improved security template mechanism for Windows.

7. **Additional Protection Mechanisms**

The template mechanism, while powerful, cannot address all the security settings that we think should be configured, as it is limited to configuring the system
registry. Below, we briefly mention additional safeguards that complement templates and yield an even higher level of system security:

- Employing SysKey to protect Security Accounts Manager (SAM) from password cracking software (e.g., ophcrack) [6]. For example, during system boot, a user will have to enter an additional password.
- Renaming the administrator account to a new, misleading name; then, creating a dummy Administrator account that is not a member of any group and has no privileges but, instead, has an extremely long and complex password [7]. This account will function as bait for hackers trying to crack it.
- Converting all drives to the New Technology File System (NTFS) and setting appropriate permissions [7].
- Also, it is worth considering to replace the built-in Windows firewall by a better solution, as the firewall exposes some disappearing drawbacks [7].

8. Conclusions and Future Work

In this paper we discuss the configuration of system security in Windows. However, we assume that the system cannot be modified. Having access to a source code (which is possible, e.g., through the Microsoft Government Security Program), a great, wide area of security improvements opens. These improvements mainly concern national requirements and may involve:

- Supplementing the login process with hardware-assisted methods, such as biometry-based user identification (who the user is) and a hardware identifier (what the user owns).
- Protecting hard drives with national-specific encryption algorithms (and, for maximum security, an external hardware key).
- Adding yet another layer of security, namely, a virtual machine manager (Windows would be run in a dedicated virtual machine).
- Implementing an advanced auditing mechanism that would be resistant to tampering and deletion attempts.

The results of our work, presented in the paper, are the following:

- Analyzing available secure templates for Windows, combining them and augmenting with additional statements.
- Proposing simple yet effective extensions to the current template syntax that greatly reduce complexity, improve readability and introduce some new features.
- Designing and implementing a .NET Framework class library for processing security templates (both standard and extended notations are supported).
- Providing an integrated tool for reading, writing, editing, comparing templates, applying them to the system and importing the current configuration to a template.
In our opinion, future work on template-based Windows security configuration should focus on augmenting our tool with two significant features:

- GUI-based template editing (which would require the ability to process the sceregvl file), including the SDDL strings. The last function would require extending the class library with a dedicated class for interpreting and comparing SDDL strings.

- The ability to convert a document using the standard syntax to the new, extended notation. This process of “compressing” cannot be completely automatic (e.g., a tool cannot generate meaningful names for constants), although it could significantly help a user editing template through finding and grouping similar entries (e.g., system services that are configured with the same SDDL strings).

REFERENCES


