monitoring of their technical condition on the major modes of technical operation.

The scientific novelty of this technique is determined by a complex use of methods of analysis of the emergency situation, predicting possible events and accidents, the method of deep rapid diagnosis and method of sampling of technical state fleet of vehicles, as well as the method of assessment of the security process operations of the maintenance and recovery vehicles.

And implementation of the proposed method will use a comprehensive approach to manage operational safety vehicle on all major modes of operation in technical units and SBGU, and develop practical recommendations to improve the system of technical operation of vehicle taking into account aspects of operational safety.

Keywords: *methods, safe operation, vehicles, management.*

УДК 681.3.068

Сергій Михайлович ЦИРУЛЬНИК,
кандидат технічних наук, доцент, завідувач кафедри «Радіотехніка» Вінницького технічного коледжу, м. Вінниця

Дмитро Олександрович ВОЛИНЕЦЬ,
старший викладач кафедри зв'язку, автоматизації та захисту інформації Національної академії Державної прикордонної служби України імені Богдана Хмельницького, м. Хмельницький

ЗАХИСТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ AVR МІКРОКОНТРОЛЕРІВ

У статті розглядаються питання захисту авторського коду у сучасних мікроконтролерах AVR від злому та небажаного дослідження. Розглянуто вбудовані засоби захисту та методи протидії автоматичним і інтерактивним дизасемблерам. Новизна запропонованих методів полягає у використанні для механізму захисту програмного забезпечення архітектури команд та адресації програмної пам'яті мікроконтролерів AVR, що дозволяє ефективно протидіяти автоматичним та інтерактивним дизасемблерам.

Ключові слова: *мікроконтролер, захист програмного забезпечення, дизасемблер, пам›ять програм.*

Постановка проблеми у загальному вигляді. При розробці сучасних електронних приладів використовуються мікроконтролери (МК). Для захисту программного забезпечення розробники МК

передбачають спеціальні заходи щодо запобігання несанкціонованого копіювання.

З точки зору рівня захисту інформації усі МК можуть бути поділені на дві групи – звичайні та захищенні. Захищенні МК мають різні рівні доступу та захисту даних на рівні кристалу з використанням кодування інформації [1]. Подолання захисту таких МК потребує використання дуже складного та дорогого обладнання, що економічно не доцільно.

Аналіз останніх досліджень і публікацій, в яких започатковано вирішення даної проблеми та на які опирається автор. Найважливішим трендом у споживчій техніці останнього часу став так званий Internet ofthings. Розумні телевізори, розетки, лампочки, машини, ваги. Проблеми, з якими раніше стикалися виробники роутерів та IP-камер, тепер стають актуальними для майже будь-яких пристроїв навколо нас. Перед кожним розробником (або групою розробників) електронної техніки рано чи пізно постає питання про захист прав на інтелектуальну власність. Формально і найскладніша програма, і найпростіша повинні бути однаково захищені як інтелектуальна власність програміста. На відміну від праці письменника, де видання невеликого оповідання дає деяку гарантію авторських прав, у програмуванні все навпаки. Досить популярні мікроконтролери AVR та платформа Arduino використовується для створення автономних об'єктів автоматики, робототехніки. У зв'язку з цим необхідно розглянути методи захисту інтелектуальної власності програміста на прикладі модифікації програм для мікроконтролерів AVR.

Метою статті є дослідження методів захисту програмного забезпечення мікроконтролерів загального використання AVR фірми Atmel від дизасемблювання, реверс – інжирінга та повторного перепрограмування.

Розглянути методи захисту програмного коду мікроконтролерів AVR від несанкціонованого зчитування, модифікації та протидії автоматичним та інтерактивним дизасемблерам.

Виклад основного матеріалу дослідження. На сьогодні AVR мікроконтролери мають вбудовані засоби захисту програмного забезпечення від несанкціонованого доступу. Реалізовані вони у вигляді Lock-бітів [2, 3]. Це група конфігураційних (fuse) бітів, яка відповідає за доступ до програмної пам'яті та EEPROM. Налаштувавши їх можна заборонити читання програмного забезпечення з мікроконтролера.

Усі мікроконтролери AVR мають два Lock-біта (LB1 та LB2), незапрограмовані за замовчуванням. Якщо запрограмувати тільки LB1, то запис Flash та EEPROM буде заборонено, а читання дозволено. Щоб заборонити ще й читання, потрібно також запрограмувати LB2 (табл. 1)

Таблиця 1

Режими захисту AVR мікроконтролерів

Режим	LB2	LB1	Опис
1	1	1	Захист вимкнено
2	1	0	Заборона запису Flash-пам'яті та EEPROM
3	0	0	Заборона читання і запису Flash-пам'яті та EEPROM

Після програмування Lock-бітів (режим 2 або 3) доступ до всіх інших конфігураційних бітів закривається, тому біти захисту слід програмувати в останню чергу.

СкинутиLock-біти можна лише виконавши повне стиранням мікроконтролера. Такий захист простий і дієвий, проте теоретично, маючи спеціальне обладнання можливо фізично втрутитись в кристал та скинути Lock-біти, знаючи їх розташування. Цей процес дуже дорогий [1] залежно від моделі мікроконтролера. Захистом від таких атак служить економічна доцільність: рідко 8-бітні мікроконтролери застосовуються в проектах такої вартості.

Існує метод обходу захисту Lock-бітів. Нагадаємо, що конфігураційні біти в запрограмованому стані мають низький рівень. Метод полягає в підвищенні напруги живлення, доки рівень запрограмованих конфігураційних бітів не перейде межу високого рівня (близько 8 В) [2]. При цьому мікроконтролер швидше за все вийде з ладу, і немає гарантії успішного зчитування прошивки мікроконтролера. Оскільки

метод дуже ризикований, то використовується він тільки у випадку, коли є велика кількість мікроконтролерів з цільовою прошивкою для експериментів.

Опираючись на існуючі методи обходу захисту, можна зробити висновок, що в не дуже дорогих проектах цілком вистачає вбудованої системи захисту AVR мікроконтролерів.

Якщо в проекті не планується подальше перепрограмування мікроконтролера, то встановивши відповідний конфігураційний біт (RSTDSBL) можна відключити вивід Reset. Проте теоретично його можна скинути наведеними вище методами, від яких знову ж таки рятує економічна доцільність.

Часто необхідно випускати оновлення програмного забезпечення, а поширювати його потрібно у відкритому доступі. Для таких цілей зазвичай перепрограмування виконують за допомогою бутлоадерів (bootloader) – програм, записаних в мікроконтролер, що приймають оновлену версію прошивки по зовнішньому інтерфейсу та записують її в програмну пам'ять (команда SPM). Бутлоадери підтримуються в усіх мікроконтролерах сімейства Mega [4].

Перевага бутлоадера в даному випадку полягає в тому, що прошивку можна зашифрувати, наприклад, застосовуючи скремблер, а розшифрується вона вже в мікроконтролері при виконанні перепрограмування.

Наведені вище методи не гарантують абсолютного захисту, тому слід припускати можливість, що hex-файл потрапить до зловмисників, який вони можуть дизасемблювати та модифікувати програму [5, 6].

Здавалось би, асемблювання – однозначне переведенням немокодів в програмний код та місця для заплутування програми тут немає. Проте завдяки особливостям команд AVR асемблера існують підходи, що ускладнюють відлагодження до такого рівня, що виникає питання доцільності відлагодження [7]. Тобто, повторне написання програми може зайняти менший час. Дизасемблери існують двох видів: автоматичні та інтерактивні [8, 9]. Автоматичні дизасемблери просто переводять машинні коди команд в мнемо коди AVR асемблера. Інтерактивні ще й будують граф виклику підпрограм, переходів, відновлюють

директиви [6, 8]. Тобто спрощують побудову алгоритму програми з її машинного коду.

В середовище розробки AVR Studio вбудований автоматичний дизасемблер [7], тому всі приклади будуть показані в ньому.

Велике поле для заплутування коду програми дають команди переходів [3, 4, 5]. Найпростішим прикладом заплутування програми переходом є розміщення в програму мовного переходу з наперед відомим результатом умови. Тобто точно відомо, що умова хибна, перехід не відбудеться, а інтерактивний дизасемблер додасть одну зайву гілку в графі переходів. Кілька зайвих гілок і час побудови алгоритму для зловмисника значно зростає.

Приклад (мікроконтролер ATmega 8):

д (мікроконтролер ATmega 8):

```
.include "m8def.inc"


RESET:
clt
ldi R16, low(RAMEND)
out SPL, R16
ldi R16, high(RAMEND)
out SPH, R16

. . . .

brts RESET  ;перехід не відбудеться, адже прапор T скинутий
```

У системі команд AVR мікроконтролерів є команда переходу за динамічною адресою (IJMP – indirect jump) [7]. Якщо реалізувати безумовний перехід з її допомогою, то аналіз коду сильно ускладнюється. Потрібно просто отримати адресу переходу з якогось нескладного математичного розрахунку (з потрібним розробнику результатом). Людині, що аналізує дизасембльований код доведеться відтворити цей розрахунок, щоб визначити адресу переходу. Приклад:

```
.include "m8def.inc"

;в регістр Z записується модифікована адреса мітки

ldi ZL, low(2*label_1+5)
ldi ZH, high(2*label_1+5)

;адреса розшифровується
subi ZL, 5
lsr ZL
ror ZH

;перехід
ijmp

;цільова мітка
.org $90C
label_1:
nop
```

Доволі просто заставити дизасемблер переплутати безумовний перехід з виходом з процедури: в стек записується адреса переходу (спочатку молодший байт) та викликається команда RET. Приклад:

```
.include "m8def.inc"

;в регістр Z записується модифікована адреса мітки

ldi ZL, low(label_1)
ldi ZH, high(label_1)

;занусуємо в стек адресу повернення з процедури і виконуєм команду повернення
push ZL
push ZH
ret

;цільова мітка
.org $90C
label_1:
nop
```

при аналізі коду, який є дизасемблованим, втрачаються точки виходу в процедуру та зрозуміти логіку стає важче.

Основуючись на архітектурі команд та адресації програмної пам'яті можна застосувати нові методи ускладнення відлагодження.

Більшість команд займають у пам'яті програм 1 слово (2 байти), проте існують команди, що займають 2 слова (4 байти). Це команди, що працюють з прямою адресацією.

Приклади:

JMP k – перехід на адресу k;

CALL k – виклик процедури за адресою k;

LDS k – завантажити в регістр значення, що знаходить за адресою k;

STS k – зберегти значення регістра в комірку з адресою k.

Серед усіх 4-байтових команд найбільшу підтримку мають LDS та STS. Вони є в усіх мікроконтролерах, що мають оперативний запам'ятовуючий пристрій (ОЗП).

Адресується пам'ять програм по словам і після виконання кожної команди вміст лічильника програми (Program Counter) інкрементується. Після виконання 4-байтових команд його вміст збільшується на 2. Якщо за допомогою команд переходів занести в нього адресу другої половини 4-байтної команди (а вона вказує на операнд k), то виконається команда, закодована адресою k. Наприклад, приховаємо команду SET:

```
.include "m8def.inc"

rjmp label_1 + 1

;...

label_1:
lds R16, $9468 ;команда SET
```

Вигляд у дизасемблері:

```
---- myProg.asm ----------------------------------------------------------------
3:          rjmp label_1 + 1
+00000000:  C001        RJMP    PC+0x0002       Relative jump
@00000001: label_1
8:          lds R16, $9468 ;команда SET
+00000001:  91009468    LDS     R16,0x9468      Load direct from data space
---- No Source -----------------------------------------------------------------
+00000003:  FFFF        ???                     Data or unknown opcode
+00000004:  FFFF        ???                     Data or unknown opcode
+00000005:  FFFF        ???                     Data or unknown opcode
+00000006:  FFFF        ???                     Data or unknown opcode
+00000007:  FFFF        ???                     Data or unknown opcode
F:\PRG\PROJECTS\AVR\SequrityTests\Source\myProg.asm   Disassembler
```

Як видно з рисунка, дизасемблер визначив команду LDS, команду SET він не бачить.

Еквівалентний код без приховання:

```
.include "m8def.inc"

rjmp label_1

;...

label_1:
set
```

Недолік методу полягає в тому, що для кожної команди, яку слід приховати необхідно шукати машинний код в довідниках. Якщо замість 4-байтної команди написати її код через директиву DW, то наступну (приховану) команду можна вказати мнемо кодом. Результат

трансляції буде таким же, але відпадає необхідність переривати довідники команд, необхідно лише вивчити код якоїсь 4-байтної команди, наприклад LDS - $9100 для регістра R16.

Перепишемо, з урахуванням цього, попередній приклад:

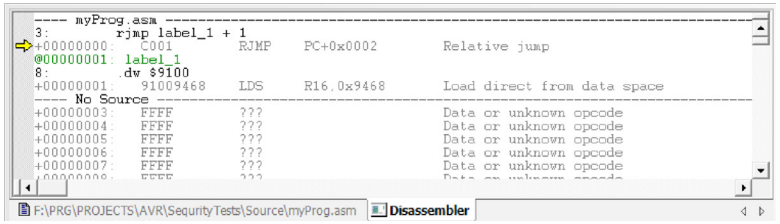
```
.include "m8def.inc"

rjmp label_1 + 1

;...

label_1:
.dw $9100 ;код команди LDS
set
```

Програму писати значно простіше, а дизасембльований код не змінився:



При довгому і уважному аналізі дизасембльованого коду усі ці методи захисту можна обійти, але це вимагає часу більше ніж на написання нової програми, що знову притягує фактор економічної доцільності.

Висновки дослідження перспективи подальших розвідок у даному напрямку. Звичайно, універсального захисту не існує. Застосувавши описані в статті методи захисту, не можна бути на сто відсотків упевненими в тому, що ніхто не запозичує ваш код. Однак витрачений на це час буде порівняний з часом написання подібної програми.

Наукова новизна запропонованих методів полягає у використанні для механізму захисту програмного забезпечення архітектури команд та адресації програмної пам'яті мікроконтролерів AVR, що дозволяє ефективно протидіяти автоматичним та інтерактивним дизасемблерам.

Практична значимість методів полягає у підвищенні рівня захисту інтелектуальної власності «embedded» програміста до економічно обґрунтованого.

Список літератури

1. Скоробогатов С.Л. Защита современных микроконтроллеров от копирования/ С. Л. Скоробогатов// НАУЧНАЯ СЕССИЯ МИФИ. – 2001. – №1. – С.84-85.

2. Цирульник С.М. Захист програмного забезпечення в AVR мікроконтролерах/ С. М. Цирульник, А.С. Зимогляд // Матеріали IV МНТК «Методи та засоби кодування, захисту й ущільнення інформації».–Вінниця, 2013

3. Шишкин С. Способ защиты программного обеспечения микроконтроллеров/ С. Шишкин // Современная электроника – 2010. – № 4. – С. 68-71.

4. Баландин Н. Антиотладочные приёмы для 8 битных микроконтроллеров AVR/ Н. Баландин// Современная электроника. – 2010. – № 1. – С. 64-66.

5. Панов А. С. Реверсинг и защита программ от взлома / А. С. Панов. – СПб.: БХВ-Петербург , 2006. –256 с. – ISBN 5-94157-889-X

6. Пирогов В. Ю. Ассемблер и дизассемблирование / Ю. В. Пирогов. – СПб.:БХВПетербург, 2006. –458 с. – ISBN 5-94157-677-3, 5-94157-677-7.

7. Ревич Ю. В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера/ Ю. В. Ревич. – СПб.: БХВ-Петербург, 2011. – 352 с. (Аппаратные средства) .– ISBN 978-5-9775-0277-1.

8. Офіційна web-сторінка компанії Apline [Електронний ресурс] / IDA Pro - самый мощный дизассемблер в мире. – Режим доступу: http://www.idasoft.ru/disassembler, вільний. – Загл. зекрана. – Моварос., англ.

9. The Online Disassembler: ODA [Електронний ресурс]. – Режим доступу: https://www.onlinedisassembler.com, вільний. – Загл. зекрана. – Мова англ.

Рецензент – доктор технічних наук, професор Андрощук О. С.

Стаття надійшла до редакції 13.11.2015

Защита програмного обеспечения AVR микроконтроллеров

В статье рассматриваются вопросы защиты авторского кода в современных микроконтроллерах AVR от взлома и нежелательного исследования. Рассмотрены встроенные средства защиты и методы противодействия автоматическим и интерактивным дисассемблерам. Новизна предлагаемых методов заключается в использовании для механизмов защиты программного обеспечения архитектуры команд и адресации программной памяти микроконтроллеров AVR, которые