Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Útoky na hardvér pomocou indukovania chýb

Bakalárska práca

2023
Dennis Vita

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Útoky na hardvér pomocou indukovania chýb

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	FMFI.KI - Katedra informatiky
Školiteľ:	RNDr. Richard Ostertág, PhD.

Bratislava, 2023
Dennis Vita

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Dennis Vita
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Útoky na hardvér pomocou indukovania chýb
Attacking hardware using fault injection

Anotácia: Pomocou externých faktorov je možné ovplyvniť správanie hardvéru až do takej miery, že nekorektne vykoná zvolenú operáciu. Toto neštandardné správanie je potom možné využiť na cielený útok na daný hardvér. Medzi tieto externé vplyvy patrí napríklad zmena napájania (obvykle podpätie) alebo elektromagnetické rušenie.

Cieľom práce je zozbierať, podrobne popísať (vrátane potrebných zapojení a nástrojov) a prakticky vyskúšať rôzne možnosti ovplyvnenia vykonávania algoritmov. Výsledkom práce by malo byť overenie ako veľmi záleží na presnosti a časovaní pri takýchto útokoch na rôzny hardvér (napríklad Arduino, STM32). Či je potrebné pri implementácii využiť rýchlosť a presnosť FPGA alebo stačí pre útok použiť lacný a menej výkonný hardvér. Ďalším cieľom práce je príprava nových úloh využívajúcich indukovanie chýb do súťaží typu Capture The Flag (CTF). Úlohy by pozostávali zo zadania, vzorového riešenia (zapojenie, program, obrázky, ...).

Vedúci: RNDr. Richard Ostertág, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 30.09.2022

Dátum schválenia: 05.10.2022 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Ďakujem vedúcemu bakalárskej práce RNDr. Richardovi Ostertágovi, PhD. za odborné vedenie, metodickú pomoc, podnetné nápady, pripomienky a konzultácie pri písaní práce. Ďakujem aj Tomášovi Stykovi za poskytnutie dodatočnej pomoci pri návrhu implementácie niektorých častí práce.

Táto práca vznikla aj vďaka podpore v rámci Operačného programu Integrovaná infraštruktúra pre projekt: Advancing University Capacity and Competence in Research, Development and Innovation (ACCORD), spolufinancovaný zo zdrojov Európskeho fondu regionálneho rozvoja.

Abstrakt

Vplyvom externých fyzikálnych faktorov (zmena napájania, elektromagnetické rušenie a pod.) je možné ovplyvniť činnosť hardvéru až do takej miery, že nekorektne vykoná niektorú operáciu. Indukovanie chýb je technika útoku, ktorá zneužíva toto správanie a možno pomocou nej špecifickým spôsobom zaútočiť na konkrétny hardvér. Pri správnom načasovaní na kritickú časť programu je možné cielene ovplyvniť daný hardvér, napríklad s účelom obísť bezpečnostný mechanizmus alebo narušiť implementáciu kryptografického algoritmu. Jedným cieľom práce je overiť, či aj lacným hardvérom možno realizovať úspešný útok. Zároveň sa v práci zaoberáme podrobnou analýzou vybraných útokov a ich možným dopadom na vybraný hardvér (mikrokontrolér ATMega328P). Cieľom je aj zistiť, aké možnosti indukovania chýb pre útočníka lacný hardvér poskytuje, napríklad vynechanie inštrukcie, ovplyvnenie výsledku operácie a pod. Výsledky analýzy na záver demonštrujeme úspešnými útokmi na vytvorené cvičné príklady programov.

Kľúčové slová: indukovanie chýb, mikrokontrolér, zmena napájania

Abstract

The influence of external physical factors (power supply voltage changes, electromagnetic interference, etc.) can affect the operation of hardware to such an extent that it may incorrectly execute an operation. Fault injection is an attack technique that exploits this behavior and can be used to attack specific hardware. By proper timing and targeting a well-chosen part of a program, it is possible to deliberately influence the targeted hardware, for example, to bypass a security mechanism or cause faults in an implementation of a cryptographic algorithm. One goal of this thesis is to verify whether successful attacks can also be performed using inexpensive hardware. Additionally, the thesis focuses on a detailed analysis of selected attacks and their potential impact on specific hardware (the ATMega328P microcontroller). The aim is also to explore what fault injection capabilities inexpensive hardware to an attacker provides, such as skipping an instruction or influencing the result of an operation.. The analysis results are demonstrated by successful attacks on simple program examples.

Keywords: fault injection, microcontroller, power glitch

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Za posledné roky výrazne vzrástlo využitie programom riadených zariadení, či už v domácnosti, produkčnej sfére alebo v štátnej, či celosvetovej infraštruktúre. Fungovanie každého programu v konečnom dôsledku zabezpečuje hardvér, ktorý na tej najnižšej úrovni jednotlivé inštrukcie programu vykonáva.

Pri návrhu a implementácii softvéru sa väčšinou predpokladá, že hardvér, na ktorý bude program nasadený bude jednotlivé inštrukcie vykonávať bezchybne a presne tak, ako výrobca daného hardvéru uvádza. Za bežných podmienok je väčšinou tento predpoklad skutočne naplnený. Vplyv externých fyzikálnych faktorov, ktoré sú mimo špecifikácie udávanej výrobcom, ako napríklad prudká a krátka zmena napätia, elektromagnetické rušenie a mnoho ďalších, však môže mať za následok chybné vykonanie aktuálnej inštrukcie a tým spôsobiť nedefinované správanie bežiaceho procesu. Takýmto spôsobom možno cielene vyvolať konkrétnu chybnú operáciu v správnom čase a zaútočiť tak na dané zariadenie, napríklad s účelom obísť bezpečnostný mechanizmus. Takýto typ útokov na hardvér sa nazýva útok indukovaním (vyvolaním) chyby (angl. fault injection attack).

Špeciálna pozornosť pri útokoch pomocou indukovania chýb je venovaná vnoreným (angl. embedded) zariadeniam, ktoré sú okrem iného používané aj na obsluhu bezpečnostných systémov ako napríklad inteligentné zámky, bezpečnostné kamery a pod. Funkčnosť veľkej časti bezpečnostných systémov závisí aj na koncových zariadeniach. Cielené ovplyvnenie ich správania môže predstavovať bezpečnostnú hrozbu.

Útokmi pomocou indukovania chýb sa zaoberá veľké množstvo článkov a prác [12, 4, 7, 5, 10, 20, 13], ktorých výsledkom sú často úspešné útoky. Náklady na realizáciu útoku pomocou indukovania chýb sa výrazne líšia. Väčšina existujúcich prác využíva pre implementáciu samotného útoku stredné až veľmi vysoké náklady (rádovo od niekoľko stoviek až po desaťtisíce eur). Hlavným cieľom tejto práce je práve overenie či aj veľmi lacný hardvér, ktorého cena sa pohybuje rádovo v desiatkach eur je použiteľný pre útoky pomocou indukovania chýb.

V kapitole 1 predstavíme základné techniky indukovania chýb aj stručne vysvetlíme ich základné princípy. V nasledujúcej kapitole 2 popíšeme kľúčové aspekty zvoleného hardvéru, ktoré budú podstatné pri implementácií jednotlivých útokoch. Kapitola 3 predstavuje hlavnú časť práce, v ktorej sa pokúsime implementovať vybrané útoky.

Zároveň sa analýzou pokúsime určiť aký typ chyby (vynechanie inštrukcie, ovplyvnenie výsledku, stavu pamäte a pod.) možno lacným hardvérom dosiahnuť a aká presnosť načasovania je pri útoku potrebná. Na záver, v kapitole 4, aplikujeme tieto znalosti a demonštrujeme úspešný útok na jednoduché programy bežiace na mikrokontroléri ATMega328P [1].

Kapitola 1

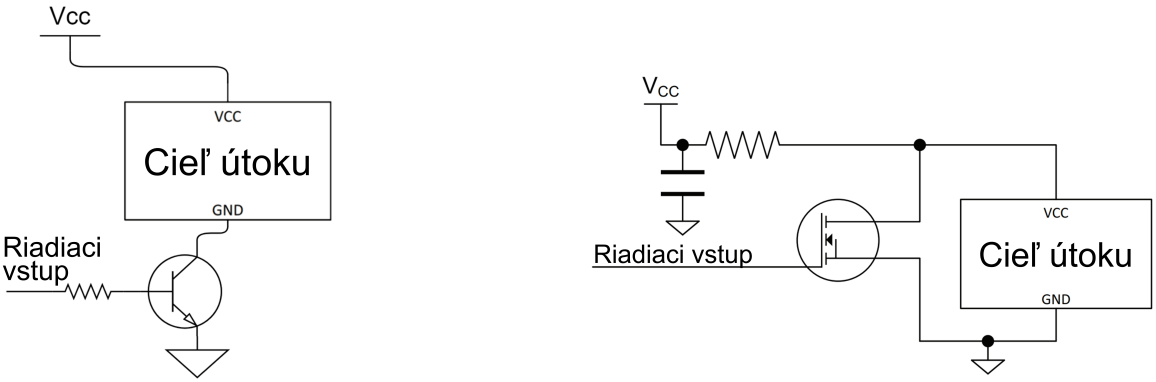
Techniky, princípy a ciele útokov

V tejto kapitole uvedieme najčastejšie techniky používané na indukovanie chýb v hardvéri a vysvetlíme princípy týchto techník. Zároveň pre niektoré útoky zhrnieme existujúce výsledky popísané v iných prácach venovaných indukovaniu chýb.

Častým predpokladom úspešného útoku pomocou indukovania chýb je fyzický prístup k zariadeniu, ktoré je cieľom útoku [7]. Väčšina takýchto útokov vyžaduje miernu modifikáciu hardvéru, aby bolo možné ovplyvniť jeho činnosť, napríklad odstránenie niektorých komponentov, ktoré by mohli útok sťažiť, prípadne úplne znemožniť. Ideálnym scenárom pre útočníka je, ak môže komponent vybrať a prevádzkovať ho vo svojom prostredí. Aj preto sú častými obeťami takýchto útokov vnorené zariadenia [7], väčšinou ovládané pomocou MCU (Micro Controller Unit, skrátene mikrokontrolér).

1.1 Zmena napätia

Jednou z najjednoduchších a veľmi často využívaných techník indukovania chýb je zmena napätia, obvykle podpätie (angl. power glitch) [12, 6, 5]. Pre viac informácií o indukovaní chýb technikou zmeny napätia odporúčame článok venujúci sa rôznym typom zmeny napätia [20]. Princíp takéhoto útoku spočíva v skonštruovaní obvodu, ktorý bude mať kontrolu nad napájaním zariadenia, na ktoré chceme útočiť a vo vhodných okamihoch na veľmi krátky čas (rádovo stovky nanosekúnd), vyvolá prudkú zmenu napätia. Takáto manipulácia môže spôsobiť nedefinované správanie na napájanom zariadení, najčastejšie možno očakávať, že ovplyvní aktívne časti hardvéru, napríklad obvody na procesore vykonávajúce jednotlivé inštrukcie. Častými symptómami takéhoto útoku sú napríklad preskočenie inštrukcie, nekorektné vyhodnotenie podmieneného skoku, chybný výsledok aritmetickej, či logickej operácie a pod. Najdôležitejšia časť takéhoto útoku je správne načasovanie a dĺžka intervalu zmeny napätia. Pokiaľ je tento interval zmeny napätia príliš dlhý (viac ako 2-3 mikrosekundy), je veľká pravdepodobnosť, že nastane fatálne zlyhanie a následný reštart zariadenia. Práve požadovaná presnosť



(a) Ovládanie tranzistorom

(b) Skratovanie zdroja

Obr. 1.1: Príklad obvodov pri útoku zmenou napätia. Obvod vľavo 1.1a ovláda napájanie cieľa pomocou tranzistora [6], vpravo 1.1b je obvod, pomocou ktorého možno skratovať napájací zdroj [12].

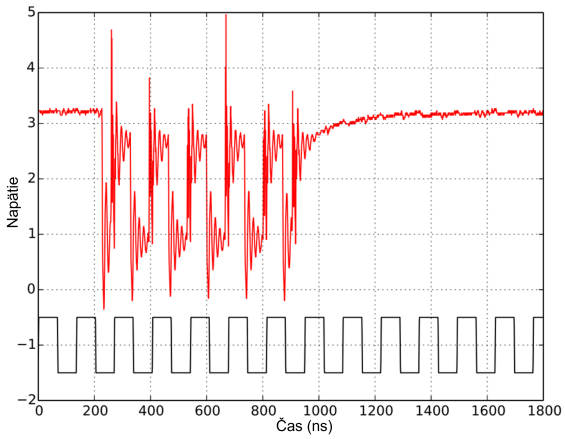
časovania je parameter, ktorý výrazne vplýva na technickú náročnosť útoku.

Pre implementáciu útoku využívajúceho techniku zmeny napätia sú zvyčajne potrebné dve základné súčasti. Jednou je obvod, ktorý dokáže dynamicky manipulovať s napájaním zariadenia, na ktoré cielime a druhým je generátor riadiacich signálov pre tento obvod, ktorý dokáže generovať impulzy s dostatočnou presnosťou. V závislosti od zložitosti a požadovaných parametrov týchto dvoch komponentov sa odvíjajú aj náklady a náročnosť takéhoto útoku. Napriek tomu v porovnaní s inými technikami zmena napätia patrí k menej náročným na implementáciu a je preto veľmi často používaná.

Jednoduchou implementáciou takéhoto útoku môže byť napríklad použitie tranzistora, ktorým vieme spínať napájanie na mikrokontroléri. Takýmto spôsobom vieme na krátky okamih zatvorením tranzistora vyvolať podpätie na mikrokontroléri a jeho následným otvorením vrátiť napätie do bežného stavu. Tým môžeme vyvolať chybné vykonanie jednej alebo viacerých nasledujúcich inštrukcií [6].

Útok založený na podobnom, ale mierne zložitejšom princípe využíva tzv. „Crowbars“ obvod. Princíp takéhoto obvodu spočíva v tom, že zdroj napájania paralelne zapojíme cez tranzistor do skratu. Generovaním impulzov do tranzistora vieme na veľmi krátky čas (desiatky až stovky nanosekúnd) vyskratovať napájací zdroj, čím spôsobíme prudké zmeny v napätí na mikrokontroléri. Aby bolo možné presnejšie zacieliť na konkrétnu časť vykonávaného kódu bolo ovládanie tohto obvodu synchronizované s externými hodinami cieľového zariadenia, znázornené na obrázku 1.2 [12]. Výhodou takéhoto zapojenia v porovnaní s odpájaním cez tranzistor je rýchlejší pokles napätia. Nevýhodou môže byť vysoké zaťaženie zdroja počas skratu a veľký prúd, ktorý potečie cez obvod vytvárajúci skrat. Na obrázku 1.1 sú pre porovnanie znázornené schémy oboch spomenutých obvodov pre manipuláciu napájania cieľového zariadenia.

Existujú rôzne spôsoby zapojenia, ktoré umožňujú meniť napätie na cieľovom za-



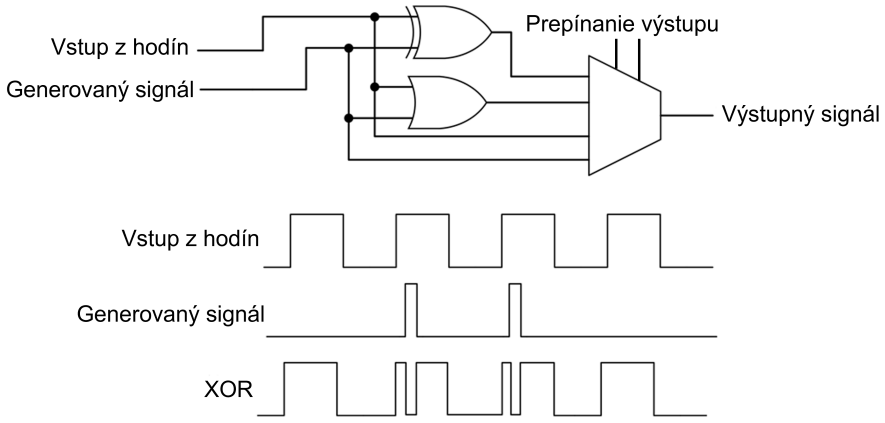
Obr. 1.2: Synchronizácia zmeny napätia s hodinami zariadenia. Červenou farbou je znázornená hodnota napätia v čase. V spodnej časti grafu (čierna farba) sú hodinové impulzy zariadenia [12].

riadení. Líšia sa typom zmeny napätia, ktorý pomocou nich dosiahneme. Citlivejšie zariadenia by na úplné odpojenie napájania mohli zareagovať reštartom. Pri útoku na takéto zariadenia preto môže byť vhodnejšie vyvolať pokles napätia nie až na 0 V, ale napríklad na polovicu napájacieho napätia. V iných prípadoch môže byť potrebné vyvolať naopak prepätie na vyššiu hodnotu ako je napájacie napätie, prípadne tzv. zvonenie (angl. ringing) – oscilovanie hodnoty napätia okolo úrovne napájacieho napätia.

Pre použitie týchto techník je dôležité zabezpečiť, exaktné časovanie generovaných riadiacich impulzov. To je možné zabezpečiť napríklad laboratórnym zdrojom alebo použitím špecializovaného hardvéru, napríklad open source platformy ChipWhisperer [14]. ChipWhisperer integruje mnoho zariadení a obvodov používaných pri indukovaní chýb do jedného zariadenia a poskytuje ucelenú, voľne dostupnú skupinu nástrojov (angl. toolchain), pre zjednodušenie a automatizáciu jeho ovládania. Takéto zariadenia však môžu vyžadovať netriviálne náklady. Otázkou je, či aj lacnejšie riešenie, napríklad mikrokontrolér ATMega328P, ktorý je súčasťou populárnych vývojových dosiek Arduino, dokáže dosiahnuť rovnako dobré alebo aspoň porovnateľné výsledky. Jedným z cieľov tejto práce je práve vyskúšať takéto typy útokov pomocou mikrokontroléra ATMega328P.

1.2 Manipulácia hodín

Ďalšou často používanou technikou indukovania chýb je manipulácia impulzov prichádzajúcich z externých hodín do zariadenia (angl. clock glitch). Takouto manipuláciou možno vytvoriť nepravidelný taktovací signál, ktorý môže spôsobiť nekorektné správanie ním riadeného hardvéru. Riadiaca jednotka procesora na zariadení obvykle reaguje na nábehovú (niekedy dobehovú) hranu signálu prichádzajúceho od hodín zariadenia.



Obr. 1.3: Schéma obvodu pre generovanie nepravidelného signálu. Vo vrchnej časti je znázornená schéma logického obvodu s multiplexorom, ktorým možno prepínať medzi výstupmi hradiel, v spodnej časti je príklad priebehu signálu na kanáloch multiplexora [11].

Napríklad pridanie hrán do takto generovaného signálu môže mať za následok, že sa začne vykonávať ďalšia inštrukcia skôr ako sa predošlá inštrukcia stihla korektne dokončiť [4]. Správnym načasovaním tejto poruchy možno spôsobiť cielený efekt na program vykonávaný týmto hardvérom. Aby bol takýto útok úspešný je potrebné, aby bola riadiaca jednotka zariadenia priamo taktovaná externým oscilátorom. Napríklad niektoré zariadenia pomocou PLL (Phase Lock Loop) obvodu odvádzajú z prichádzajúceho externého signálu interný [17], ktorý má rádovo vyššiu frekvenciu. Tento interný signál je následne použitý ako hodiny pre taktovanie procesora. Proti takýmto zariadeniam preto útok touto technikou pravdepodobne nebude účinný [12].

Útok manipuláciou hodín môže byť realizovaný napríklad skonštruovaním kombinačného obvodu, ktorý pomocou logických hradiel, napríklad AND, OR a XOR, skladá rôzne výstupné signály zo vstupných. Vstupom do takéhoto obvodu môžu byť rôzne zdroje impulzov napríklad oscilátory s rôznymi frekvenciami, laboratórne zdroje, generátory signálov a ďalšie. Cieľom je rôznymi spôsobmi modulovať výstupný taktovací signál tak, aby v želaných okamihoch mal nesprávny tvar (nepravidelné takty, nesprávne hrany, vyššia frekvencia) a spôsobil tak poruchu na cieľovom zariadení. Pomocou logických hradiel možno vytvoriť rôzne vzorky signálu a následne napríklad využitím multiplexora automatizovane prepínať medzi jednotlivými výstupmi a tým dynamicky meniť výstupný hodinový signál. Na obrázku 1.3 je znázornená ukážka schémy takéhoto obvodu [11]. Takýto obvod dokonca často nie je potrebné skonštruovať fyzicky, ale možno použiť aj programovateľné hradlové pole (FPGA), čo značne zjednoduší implementáciu útoku.

Poruchy, ktoré vieme touto technikou spôsobiť môžu byť rôzne, ale najčastejšie sa týkajú toku riadenia a dát, keďže hodiny generujú signál pre činnosť riadiacej jednotky

procesora. Príkladmi takýchto porúch sú vynechanie inštrukcie, nekorektné načítanie dát z pamäte alebo nekorektný zápis do registra PC (Program Counter), čo môže spôsobiť nesprávny skok v rámci vykonávaného programu. Konkrétna implementácia takéhoto útoku na AVR mikrokontrolér z rodiny ATMega, do ktorej patrí aj nami zvolený ATMega328P (bližšie popísaný v kapitole 2, sekcii 2.1) ukázala, že možno týmto spôsobom vyvolať viaceré z vyššie spomenutých porúch [4].

1.3 Elektromagnetické rušenie

Treťou známou technikou indukovania chýb je elektromagnetické rušenie alebo EMFI (Electromagnetic Fault Injection). Vplyvom elektromagnetického poľa možno narúšať fungovanie hardvéru na cieľovom zariadení a tým ovplyvňovať jeho činnosť. Takýto typ útokov zvyčajne vyžaduje dosku riadenú počítačom pohyblivú vo všetkých troch osiach (XYZ), pričom musí byť schopná s dostatočnou presnosťou nastaviť pozíciu antény, ktorá je zdrojom elektromagnetického rušenia. Generovaním impulzov do tejto antény v daných časových okamihoch možno spôsobovať želané rušenie [10].

Výhodou tejto techniky je možnosť s istou presnosťou (závisí od parametrov použitej elektroniky) útočiť na konkrétne časti obvodov na cieľovom čipe, napríklad pamäť, registre, či ALU. Nevýhodou je väčšia technická náročnosť útoku a obvykle drahší hardvér.

Sú známe aj ďalšie techniky indukovania chýb na hardvéri (napríklad laserové vypaľovanie), ktoré dokážu byť oveľa presnejšie a spoľahlivejšie. Ich implementácia je však oveľa náročnejšia, vyžaduje vysokú úroveň technickej zručnosti a veľmi vysoké finančné náklady [15]. Analýza týchto techník preto presahuje rámec tejto práce a nebudeme sa im podrobnejšie venovať.

1.4 Ciele útokov

Najčastejším cieľom útokov pomocou indukovania chýb je obídenie bezpečnostného mechanizmu. Príkladom je ovplyvnenie jednoduchej kontroly podmienky v programe vyvolaním chyby, ktorá spôsobí skočenie programu do nesprávnej vetvy, čo má efekt nesprávneho vyhodnotenia tejto podmienky. Zložitejšie útoky môžu cieliť na nesprávne vykonanie aritmetickej, či logickej operácie alebo nekorektné prečítanie dát z pamäte a následne chybné rozhodnutie algoritmu, čo spôsobí cielené obídenie daného bezpečnostného mechanizmu. Cieľom takéhoto útoku môže byť získanie neoprávneného prístupu k službám alebo dátam zariadenia. Konkrétne napríklad prečítanie obsahu pamäte s firmvérom aj pri zapnutej ochrane pred čítaním, útok na zavádzací softvér počítača, aby naštartoval systém z nepovoleného média a množstvo ďalších scená-

rov [13, 18].

Ďalšia kategória cieľov týchto útokov sú zariadenia implementujúce kryptografické algoritmy. Tie sú často veľmi náchylné na implementačné chyby a preto dopady takýchto útokov môžu byť fatálne. Pomocou indukovania chýb možno napríklad prinútiť zariadenie, aby použilo nulový vektor ako kľúč pre symetrickú šifru. Potom už je jednoduché takto zašifrované dáta priamo dešifrovať. O niečo sofistikovanejšia, ale o to účinnejšia, metóda útokov na kryptografické konštrukcie je tzv. DFA (Differential Fault Analysis). Princíp tejto metódy spočíva v indukovaní chýb v kryptografickom algoritme. Následne sa porovnávaním korektných a nekorektných výstupov určí aké inštrukcie a dáta sa v zariadení spracovávali. Takouto analýzou je možné napríklad zistiť použitý kľúč v symetrickej šifre alebo dokonca efektívne faktorizovať verejný modulus inštancie RSA schémy. Praktická úspešnosť týchto útokov bola demonštrovaná proti implementáciám schém AES a RSA na vnorených zariadeniach [5].

1.5 Zraniteľnosti a obranné mechanizmy

Samotná implementácia a následná úspešnosť útoku často závisí aj od samotného algoritmu, na ktorý je útok cielený, a spôsobu jeho implementácie od čoho sa odvíja ako veľmi môže byť náročná technická realizácia útoku. Niekedy nie je pre dosiahnutie cieleného efektu potrebné mieriť na jednu konkrétnu chúlostivú operáciu, ale stačí pokiaľ je výsledok série operácií nesprávny. Príkladom kódu, ktorý je náchylný voči chybám spôsobeným vplyvom prostredia je postupné inkrementovanie premennej v rámci cyklu, alebo séria viacerých priradení do rovnakej premennej počas výpočtu [12]. Kedy presne nastane chyba neovplyvní výsledný efekt útoku, za predpokladu, že chyba nespôsobí fatálne zlyhanie cieľového zariadenia. Pre tento typ útokov je často postačujúci lacný hardvér a nie je väčšinou potrebná synchronizácia zdroja chýb a cieľa [12, 6].

V iných prípadoch je potrebné cieliť na konkrétnu operáciu, čo zvyčajne vyžaduje vyššiu presnosť a synchronizáciu zariadení počas útoku. Príkladom takejto zraniteľnosti je zneužitie jednoduchej optimalizácie kompilátora [13]. Kompilátor pri preklade cyklu, v ktorom sa postupne dekrementovala iterovaná premenná a postupne čítal vybraný úsek pamäte, použil inštrukciu BNE (Branch if Not Equal) miesto BLT (Branch if Less Than). Takáto optimalizácia je z teoretického pohľadu správna, keďže nemení význam algoritmu a test na nulu možno efektívnejšie vykonať ako všeobecné porovnanie. Keď v takto upravenom programe spôsobíme vynechanie tejto inštrukcie práve vtedy, keď má dôjsť k ukončeniu cyklu, dosiahneme tým, že takýto program bude pokračovať v cykle až kým nepretečie hodnota iterovanej premennej. Takýto útok bol demonštrovaný a autorovi sa podarilo prečítať firmvér na mikrokontroléri napriek tomu, že k nemu nemal mať prístup. V prípade, že by kompilátor použil inštrukciu

BLT, bolo by potrebné pravidelne v každej iterácií spôsobiť takúto chybu, čo je pre útočníka náročnejšie [13].

Existuje viacero obranných mechanizmov, ktoré dokážu takéto útoky skomplikovať, niekedy až znemožniť. Medzi aktívne mechanizmy patrí detekcia takýchto útokov s využitím špecializovaného hardvéru, alebo softvéru a následne spustiť bezpečnostné opatrenie, napríklad reštart zariadenia. Aktívna detekcia útokov zvyšuje spotrebu zariadenia, čo najmä v prípade vnorených systémov môže byť hlavne pri napájaní z batérie problematické riešenie. Jednoduchší, ale niekedy postačujúci, je pasívny spôsob obrany, napríklad úprava kódu tak, aby bol menej chúlostivý voči chybám [13]. Pridanie náhodných oneskorení do programu môže sťažiť situáciu útočníkovi, ktorý potrebuje presne načasovať útok na konkrétnu operáciu [13]. Ďalšími opatreniami môže byť nepoužívanie binárnych premenných pri vetvení programu a rovnako nepoužívanie triviálnych konštánt (0, súvislý vektor jednotiek). Nastavenie konkrétnej netriviálnej hodnoty (napríklad 0xAA, 0xB9) vplyvom chyby je náročnejšie, niekedy zanedbateľne nepravdepodobné a možno tak takémuto útoku zabrániť.

Kapitola 2

Hardvér

V tejto kapitole popíšeme a rozoberieme niektoré aspekty zvoleného hardvéru, ktoré sú podstatné pre účely tejto práce.

Pri útokoch pomocou indukovania chýb je často potrebná vysoká presnosť časovania v stovkách až desiatkach nanosekúnd. Lacnejší hardvér často takúto presnosť nie je schopný dosiahnuť. Hlavným cieľom tejto práce je overiť túto skutočnosť a zistiť akú presnosť časovania dokáže takýto hardvér poskytnúť. Finančné náklady na všetky súčiastky použité pri implementovaných útokoch sa pohybujú rádovo v desiatkach eur.

2.1 ATMega328P

Najdôležitejším zariadením pre účely tejto práce je Atmel mikrokontrolér s 8-bitovou architektúrou AVR, dnes pod firmou Microchip Technology. Tento mikrokontrolér je súčasťou viacerých modelov vývojových dosiek Arduino, ktoré sú využívané pri vývoji jednoduchých vnorených zariadení. Zároveň zvykne byť použitý aj pri nasadení skutočných produktov, najmä vnorených systémov s nízkou spotrebou. Mikrokontrolér ATMega328P bol aj preto zvolený ako cieľ útokov implementovaných v tejto práci, ale hlavným dôvodom bola jeho veľmi nízka cena pohybujúca sa okolo desať eur. Techniky útokov použité v tejto práci môžu s nezanedbateľnou pravdepodobnosťou trvalo poškodiť cielený hardvér, preto sme sa rozhodli použiť viacej rovnakých čipov. Konkrétne boli testované štyri exempláre formátu ATMega328P-PU z dvoch (po dvojiciach) rôznych sérií výroby (2139E4A a 2128BQY). Okrem toho bol použitý aj ako zdroj generujúci riadiace impulzy do rôznych obvodov využívajúcich niektoré techniky útokov spomenuté v kapitole 1. Pre tento druhý účel bol použitý ako súčasť dosky Arduino Nano.

2.1.1 Kľúčové vlastnosti

Zvolený formát čipu ATMega328P-PU je v puzdre THT (Through Hole Technology), čo znamená, že vonkajšie piny na čipe sú vyvedené vo forme „kolíkov“ a je možné čip zapojiť bez potreby spájkovania pomocou bez-spájkového kontaktného poľa (angl. solderless breadboard). Takéto zapojenie potom možno jednoducho a bezprostredne modifikovať počas experimentovania. Ďalšou vlastnosťou je možnosť pripojenia externého oscilátora s frekvenciou maximálne 16 MHz, čo znamená väčšiu flexibilitu nastaviteľných parametrov jednotlivých útokov. Na čipe sa nachádza aj interný oscilátor s frekvenciou 8 MHz a možno ho využiť namiesto externého oscilátora [1].

Čo sa týka softvéru, pre programovanie mikrokontroléra ATMega328P je potrebný kompilátor (najčastejšie jazyka C) pre architektúru AVR. Nahratie programu do flash-pamäte na mikrokontroléri možno zabezpečiť napríklad pomocou ISP (In-System Programming) programátora. Pre tento účel, je potrebné, aby na mikrokontroléri bol prítomný zavádzací program (angl. bootloader), ďalej len zavádzač, ktorý dokáže komunikovať s ISP programátorom. Jednoduchý a automatizovaný spôsob pre vykonanie týchto operácií poskytuje aj integrované vývojové prostredie (IDE) Arduino IDE, ktoré sme sa rozhodli použiť. Arduino IDE interne používa kompilátor AVR-GCC a ISP programátor AVRDUDE [3], ktoré sú voľne dostupné aj samostatne.

2.1.2 Architektúra AVR

Mikrokontrolér ATMega328P implementuje 8-bitovú architektúru AVR. Niektoré aspekty tejto architektúry sú dôležité pre pochopenie niektorých častí kódu písaných v jazyku symbolických adries, ďalej len asembler, v rámci tejto práce, preto ich stručne opíšeme. Architektúra AVR je zaraďovaná do triedy RISC (Reduced Instruction Set Computer) architektúr [2]. Poskytuje tridsaťdva (R0 – R31) 8-bitových všeobecných registrov, z nich niektoré so špeciálnym významom pri vybraných inštrukciách. V tejto práci používame tieto registre len v základnom, všeobecnom význame. Pre prístup do pamäte slúžia dve základné inštrukcie LD (Load) a ST (Store) [2].

Ovládanie GPIO (General Purpose Input Output) pinov na mikrokontroléri je zabezpečené pomocou špeciálnych vstupno-výstupných registrov. Každý pin je súčasťou jedeného zo štyroch tzv. portov (ozn. A – D) a v rámci portu má priradený jeden z ôsmich bitov (0 – 7). Každý pin je potom ovládaný troma registrami DDRX, PINX a PORTX, kde X je písmeno označujúce port priradený pinu. Bit pinu potom určuje index konkrétneho bitu (0 označuje najmenej významný bit) v týchto troch registroch (rovnaký index pre každý register), ktorým možno ovládať daný pin. Význam jednotlivých bitoch v registroch je potom nasledovný: DDR (Data Direction register) určuje smer toku dát, pokiaľ je príslušný bit v tomto registri nastavený na 0, pin je vo vstupnom móde, pokiaľ je nastavený na 1, pin je vo výstupnom móde. Register PIN má

Algoritmus 2.1: Ovládanie vstupno-výstupných registrov na architektúre AVR.

```
; nastavenie bitu 5 na 1 v registri DDRB
sbi 0x04, 5

; nastavenie bitu 5 na 1 v registri PORTB
sbi 0x05, 5

; nastavenie bitu 5 na 0 v registri PORTB
cbi 0x05, 5
```

význam vo vstupnom móde a v danom bite udržiava logickú hodnotu vstupu na danom pine. Register PORT má naopak hlavný význam vo výstupnom móde a hodnota v príslušnom bite je výstupnou logickou hodnotou na priradenom pine. Tieto vstupno-výstupné registre možno adresovať priamo pomocou vyhradeného adresného priestoru v pamäti (memory-mapped I/O), alebo s využitím špeciálnych inštrukcií (port-mapped I/O).

Pre lepšie porozumenie uvedieme príklad. Chceme nastaviť digitálny pin 13 ako výstupný a na výstupe chceme periodicky prepínať medzi logickými hodnotami 0 a 1, napríklad za účelom blikať LED diódou, ďalej len LED (Light Emitting Diode). GPIO pin 13 má priradený port B, bit 5. Potrebujeme preto nastaviť v registri DDRB bit 5 na 1, čím nastavíme pin do výstupného módu. Následné prepínaním medzi hodnotami 0 a 1 v registri PORTB (bit 5) dosiahneme želaný efekt na GPIO pine 13. Nastavenie hodnoty konkrétneho bitu vo vstupno-výstupnom registri možno docieliť napríklad pomocou vstupno-výstupných inštrukcií SBI (Set Bit in I/O Register) a CBI (Clear Bit in I/O register), ktoré nastavia daný bit na 1 resp. 0 [2]. Tieto inštrukcie používajú vstupno-výstupné adresy daných registrov, ktoré v našom prípade sú 0x04 (DDRB) a 0x05 (PORTB) [1]. Ukážky jednotlivých inštrukcií v jazyku asembler sa nachádzajú v algoritme 2.1. V prípade, že by sme chceli hodnoty v registroch nastaviť pomocou inštrukcií LD a ST (prípadne ich variantov), použili by sme adresy týchto registrov v mape pamäti – 0x24, resp. 0x25 [1].

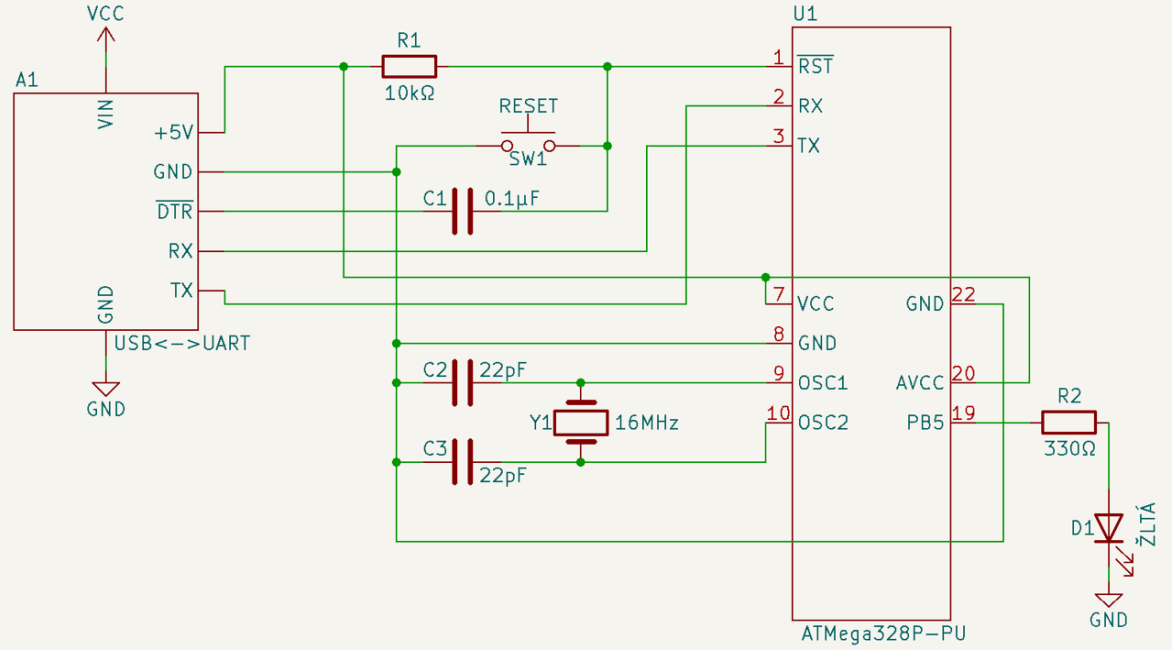
2.1.3 Základné zapojenie mikrokontroléra

V tejto časti uvedieme zapojenie mikrokontroléra ATMega328P-PU, ktoré neskôr využijeme pri implementácií vybraných útokov v kapitole 3. Zapojenie spočíva v pripojení súčiastok, ktoré zabezpečujú základné funkcie mikrokontroléra, konkrétne pripojíme zdroj napätia, externý oscilátor, tlačidlo pre hardvérový reštart, modul na konverziu medzi protokolmi USB (Universal Serial Bus) a UART (Universal Asynchronous

Receiver-Transmitter) pre komunikáciu s počítačom a LED k portu B, bitu 5, ktorú používa zavádzač na signalizáciu prebiehajúcej komunikácie. Pripojenú LED možno zároveň využiť na výstupné signály aj v samotnom programe nahratom na mikrokontroléri. Zoznam a popis všetkých použitých súčiastok:

- ATMega328P-PU – THT puzdro, umožňuje použiť kontaktné bez-spájkové pole
- prevodník medzi USB a UART – použitý modul s čipom CP2102, na doske je vyvedený aj výstup z napájania pomocou USB
- kontaktné bez-spájkové pole – umožňuje flexibilne prepájať súčiastky bez potreby spájkovania
- spínacie tlačidlo – po stlačení uzemní RST pin, čo vynúti reštart mikrokontroléra
- 16 MHz kryštál – pre zapojenie externého oscilátora
- keramické kondenzátory – 2-krát 22 pF záťažové kondenzátory ku kryštálu, 1-krát 0.1 µF pre využitie DTR pinu
- rezistory – 1-krát 330 Ω k LED, 1-krát 10k Ω – zdvíhací (angl. pull-up) odpor RST pinu
- LED dióda – použili sme žltú farbu a veľkosť 3 mm
- prepojovacie káblíky typu M-M (Male to Male)

Súčiastky zapojíme podľa schémy na obrázku 2.1. Prevodník medzi USB a UART možno okrem bežnej sériovej komunikácie použiť aj na naprogramovanie mikrokontroléra pomocou ISP programátora, za predpokladu, že mikrokontrolér obsahuje zavádzač. Pre tento účel pripájame k RST pinu na mikrokontroléri cez kondenzátor pin DTR (Data Terminal Ready) na prevodníku. To umožní automaticky vyvolať reštart na mikrokontroléri, čo je potrebné vykonať vždy pred nahratím nového programu na mikrokontrolér. Po opätovnom zapnutí sa na krátky čas (približne sekundu) spustí zavádzač na mikrokontroléri, čo umožňuje komunikáciu s ISP programátorom. Komunikáciu so zavádzačom je potrebné iniciovať v rámci tohto časového okna, v opačnom prípade zavádzač spustí program nahratý vo flash-pamäti mikrokontroléra a bude potrebné pre komunikáciu so zavádzačom mikrokontrolér opätovne reštartovať. Zvolené vývojové prostredie Arduino IDE nám umožňuje automatizovane využívať túto funkcionalitu.



Obr. 2.1: Schéma zapojenia mikrokontroléra ATMega328P.

2.2 STM32

Ďalším zariadením, ktoré sme sa rozhodli použiť je mikrokontrolér z rodiny STM32, konkrétne model STM32F407, ktorý je súčasťou vývojovej dosky STM32 F4 Discovery. STM32 má v porovnaní s ATMega328P viacero výhod. Najpodstatnejším rozdielom pre účely tejto práce je väčšia frekvencia procesora, ktorá je dokonca nastaviteľná interným PLL obvodom. Maximálna hodnota je 168 MHz, čo je značne väčšia frekvencia oproti 16 MHz na ATMega328P. Dosku STM32 F4 Discovery sme sa preto rozhodli použiť ako alternatívu k Arduino Nano, pre účel generovania riadiacich impulzov do obvodov indukujúcich chyby na cieľovom zariadení. Pri použití STM32 preto očakávame väčšiu presnosť pri načasovaní útoku. Ďalšími výhodami sú napríklad väčší výkon procesora, viac vstupno-výstupných pinov a väčšia kapacita pamäte, za cenu o niečo väčšej spotreby. Tieto aspekty však nie sú významné pre cieľ tejto práce.

Dosku STM32 F4 Discovery sme sa rozhodli programovať pomocou integrovaného vývojového prostredia STM32CubeIDE, ktoré je odporúčané aj výrobcom dosky. Prostredie poskytuje viacero užitočných funkcionalít. Podstatné pre nás bolo grafické rozhranie pre konfiguráciu vstupu a výstupu na mikrokontroléri a automatické generovanie kódu, ktorý túto konfiguráciu pri inicializácií nastaví. Takýto prístup značne zjednodušil konfiguráciu vstupno-výstupného hardvéru, ktorá je na mikrokontroléroch STM32 zložitejšia v porovnaní s ATMega328P.

2.2.1 Architektúra STM32 Cortex-M4

Mikrokontroléry STM32 sú založené na 32-bitovej Arm Cortex-M RISC architektúre. Nami použitý STM32F407 implementuje architektúru STM32 Cortex-M4. Predstavíme niektoré aspekty tejto architektúry a rozdiely z architektúrou AVR podstatné pre túto prácu. K dispozícii je trinásť 32-bitových všeobecných registrov (R0–R12). Registre R13 (SP – Stack Pointer), R14 (LR – Link Register) majú špeciálny význam pri volaní procedúr [16] a nebudeme ich v tejto práci používať. Register R15 (PC) obsahuje adresu nasledovnej inštrukcie. Pre prístup do pamäte možno použiť inštrukcie LDR (Load) a STR (Store), podobne ako v architektúre AVR. Prípadne možno použiť aj varianty ako ADR (Load PC-relative address), LDM (Load multiple registers), STM (Store multiple registers) a ďalšie [16].

GPIO piny možno obdobne ovládať pomocou vstupno-výstupných registrov, pričom v porovnaní s architektúrou AVR je ich ovládanie mierne odlišné. Piny majú opäť priradený port, tentokrát z väčšieho rozsahu (ozn. GPIOA–GPIOK), a bit priradený z rozsahu 0 – 15. Všetky vstupno-výstupné registre sú 32-bitové, v niektorých má pin priradené dva bity, ktoré môžu mať rôzny význam [17]. Týchto registrov, ktoré ovládajú piny priradené do jedného GPIO portu je v porovnaní s AVR architektúrou výrazne viac, predstavíme preto len tie, ktoré sme v práci priamo použili. Register GPIOX_IDR (Input Data register), kde X je písmeno portu (A–K), má význam vo vstupnom móde pinu. Spodné bity (0–15) obsahujú logickú hodnotu na vstupe príslušných pinov priradeným týmto bitom. Vrchné bity (16–31) v týchto registroch sú rezervované a musia mať nastavenú hodnotu 0 [17]. Register GPIOX_BSRR (Bit Set/Reset Register) možno použiť pre atomické nastavenie logickej hodnoty na výstupnom pine. Register je určený iba pre zápis (čítanie vráti vždy 0x0000). Nastavenie príslušného bitu na 1 v spodnej časti registra (bity 0–15) spôsobí nasledovné nastavenie logickej hodnoty 1 na výstupe. Naopak nastavenie bitu na 1 vo vrchnej časti registra (bit 16–31 ovláda pin s priradeným bitom 0–15) spôsobí nastavenie logickej hodnoty 0 na výstupe. Nastavenie ktoréhokoľvek bitu na 0 v tomto registri nemá žiaden vplyv na stav GPIO pinu. Piny vo Výstupnom móde možno ovládať aj pomocou ďalších registrov. Výhoda spomenutého prístupu je možnosť atomického nastavenia viacerých pinov naraz po zápise hodnoty do registra GPIOX_BSRR. Pre ovládanie a konfiguráciu GPIO pinov v architektúre STM32 CORTEX-M4 je potrebné nastaviť ďalšie vstupno-výstupné registre, tie sme však v tejto práci nenastavovali priamo na úrovni jazyka asembler, ale použili sme funkciu prostredia STM32CubeIDE, ktoré dokáže automatizovane vygenerovať potrebný kód v jazyku C, preto ich nebudeme uvádzať.

Adresovanie vstupno-výstupných portov v tejto architektúre je možné len prostredníctvom im vyhradených adries v pamäti, pomocou inštrukcií LDR a STR. Tento pamäťový priestor je organizovaný podľa portov. Každý port (GPIOA–GPIOK) začína

Algoritmus 2.2: Nastavenie hodnoty výstupného pinu GPIOD 9 na STM32F407 v jazyku asembler. Pre uloženie bázovej adresy portu GPIOD použijeme blok literálov.

```
; uloženie bázovej adresy GPIOD do r0 pomocou bloku literálov
LDR r0, =#40020C00

; nastavenie pinu na logickú 1 pomocou GPIOD_BSRR (posun 0x18)
MOV r1, #0x200
STR r1, [r0, #0x18]

; nastavenie pinu na logickú 0 pomocou GPIOD_BSRR (posun 0x18)
MOV r1, #0x2000000
STR r1, [r0, #0x18]
```

na pevnej bázovej adrese a každý typ GPIO registrov má určený posun (angl. offset) od tejto adresy. Adresa konkrétneho GPIO registra v rámci portu je potom súčet bázovej adresy portu a posunu typu registra. Niekedy je vhodné uložiť hodnotu bázovej adresy do všeobecného registra. S využitím tohto registra sa potom môžeme nepriamou adresáciou s doplnkom (pomocou inštrukcie LDR) odkazovať na jednotlivé registre daného portu. Uloženie konštanty nie je vždy možné pomocou inštrukcie MOV, keďže tá obmedzuje možné hodnoty konštánt, ktoré možno použiť v druhom operande [16]. Dôvodom obmedzenia je počet bitov (dvanásť) vyhradený pre zakódovanie konštanty v strojovom kóde. Alternatívny spôsob pre uloženie ľubovoľnej 32-bitovej konštanty do registra bez obmedzení je využitie takzvaného bloku literálov (angl. literal pool), pomocou pseudoinštrukcie asemblera. Ten takúto inštrukciu nahradí alokovaním špeciálneho priestoru v pamäti medzi inštrukciami na vhodnom mieste, na ktoré uloží túto konštantu. Následne pomocou inštrukcie LDR a relatívneho adresného módu k registru PC uloží konštantu z tohto miesta do registra, ktorý je operandom pseudoinštrukcie. V algoritme 2.2 uvádzame príklad použitia takejto pseudoinštrukcie a ovládanie výstupnej hodnoty na pine pomocou GPIO portov.

2.3 Ďalšie zariadenia a pomocné súčiastky

Okrem spomenutých mikrokontrolérov sme použili aj ďalší podporný hardvér, ktorý v tejto časti stručne opíšeme.

2.3.1 Osciloskop MDO4104C

Dôležitým pomocným zariadením, ktoré sme v tejto práci použili je laboratórny osciloskop MDO4104C. Primárnym použitím tohto osciloskopu bola podrobná analýza úrovne napätia medzi významnými časťami zapojenia počas priebehu útoku. Vďaka tomu bolo možné pozorovať stav zariadení počas útoku a následne tomu prispôsobovať útočiaci hardvér, prípadne aj časti softvéru. Osciloskop dokáže merať priebeh napätia v čase s pomerne vysokou presnosťou (desiatky až stovky nanosekúnd), čo nám pomohlo pri určení presnosti načasovania, ktorú sa počas útokov podarilo dosiahnuť. Zároveň poskytuje funkciu automaticky počítať priemer z viacerých meraní za sebou (počet je nastaviteľný). Následne zobrazí priemerný výsledok meraní, ktorý dynamicky aktualizuje ďalšími meraniami.

2.3.2 Bipolárne (BJT) tranzistory

Tranzistory sú vo všeobecnosti základným stavebným kameňom všetkých integrovaných obvodov v elektronike. Bipolárny tranzistor (BJT, angl. Bipolar Junction Transistor) má tri elektródy, bázu (B), emitor (E) a kolektor (C). Pomocou prúdu medzi bázou a emitorom možno ovládať prúd medzi kolektorom a emitorom. Keď medzi bázu a emitor privedieme dostatočné napätie na otvorenie PN prechodu (závisí od použitého tranzistora, zvyčajne okolo 0.6 V), tranzistor sa otvorí a medzi kolektorom a emitorom môže tiecť prúd. Tento princíp v práci využijeme na útok technikou zmeny napätia. Pomocou tranzistora budeme ovládať napájanie na mikrokontroléri ATMega328P využitím výstupného pinu ďalšieho mikrokontroléra pripojeného na bázu tranzistora.

2.3.3 Hradlový ovládač TC4420

Ďalší hardvérový komponent, ktorého využitie na útoky sme analyzovali je hradlový ovládač TC4420. Hradlový ovládač (angl. gate driver) je integrovaný obvod, pomocou ktorého možno riadiacimi signálmi prepínať napätie na iných súčiastkach v obvode. Túto funkcionalitu preto použijeme (podobne ako v prípade tranzistora) na útok zmenou napätia tak, že budeme pomocou tohto obvodu spínať a odopínať napätie na cieľovom ATMega328P. Keďže integrovaný obvod je kompaktný a obsahuje vnútri všetky potrebné súčiastky, jeho zapojenie je priamočiare a nevyžaduje žiadne ďalšie komponenty.

Kapitola 3

Implementácia a analýza vybraných útokov

V tejto kapitole podrobne popíšeme vybrané útoky na konkrétny hardvér, ktoré sme sa pokúsili implementovať. Zároveň sme niektoré útoky aj podrobnejšie analyzovali a následne sme sa ich pokúsili vylepšiť z hľadiska hardvéru aj softvéru. Naším cieľom bolo najmä overiť technickú náročnosť realizácie vybraných útokov a zistiť, či aj nami zvolený lacný hardvér je na takéto útoky použiteľný. Pri niektorých zapojeniach sme sa pokúsili aj presnejšie určiť (experimentálne) ako vieme ovplyvniť beh cieľového algoritmu na úrovni inštrukcií v jazyku asembler. Zdrojové kódy programov použitých na riadenie jednotlivých útokov ako aj programov, na ktoré sme útočili sa nachádzajú v elektronickej prílohe priloženej k práci.

3.1 Útok na firmvér zo súťaže CTF

Ako prvé sme sa pokúsili zreprodukovať popísaný útok na firmvér, ktorý bol realizovaný v rámci súťaže CTF (Capture The Flag) [6]. Program po spustení periodicky posiela cez rozhranie UART správu „Lock“. Po úspešnom útoku by mal poslať „tajný flag“, ktorého obsah je cieľom tejto súťaže. Potrebný hardvér pre realizáciu tohto útoku je nasledovný:

- Arduino UNO R3 – použili sme klon, obsahuje vyberateľný čip ATMega328P v THT puzdre
- Arduino Nano – taktiež klon, použité na ovládanie tranzistora ako zdroj na indukovanie chýb
- kontaktné bez-spájkové pole
- bipolárny tranzistor NPN – použili sme model 2N2222A v THT puzdre

- rezistory – použili sme 2-krát 330 Ω, (potrebný odpor sa môže líšiť v závislosti od použitého tranzistora)
- prepojovacie káblíky typu M-M

Čo sa týka softvéru je potrebný ISP programátor, ktorý dokáže nahrať firmvér na mikrokontrolér ATMega328P a kompilátor jazyka C pre architektúru AVR. Použili sme integrované vývojové prostredie (IDE) Arduino IDE, ktoré poskytuje automatizovaný spôsob kompilovania a nahrávania firmvéru, podporujúce aj náš ATMega328P. Firmvér, na ktorý chceme útočiť bol zverejnený priamo v skompilovanom HEX (Hexadecimal) obraze strojového kódu a pre nahranie takéhoto kódu sme použili open source projekt AVRDUDE [3], ktorý interne používa aj Arduino IDE.

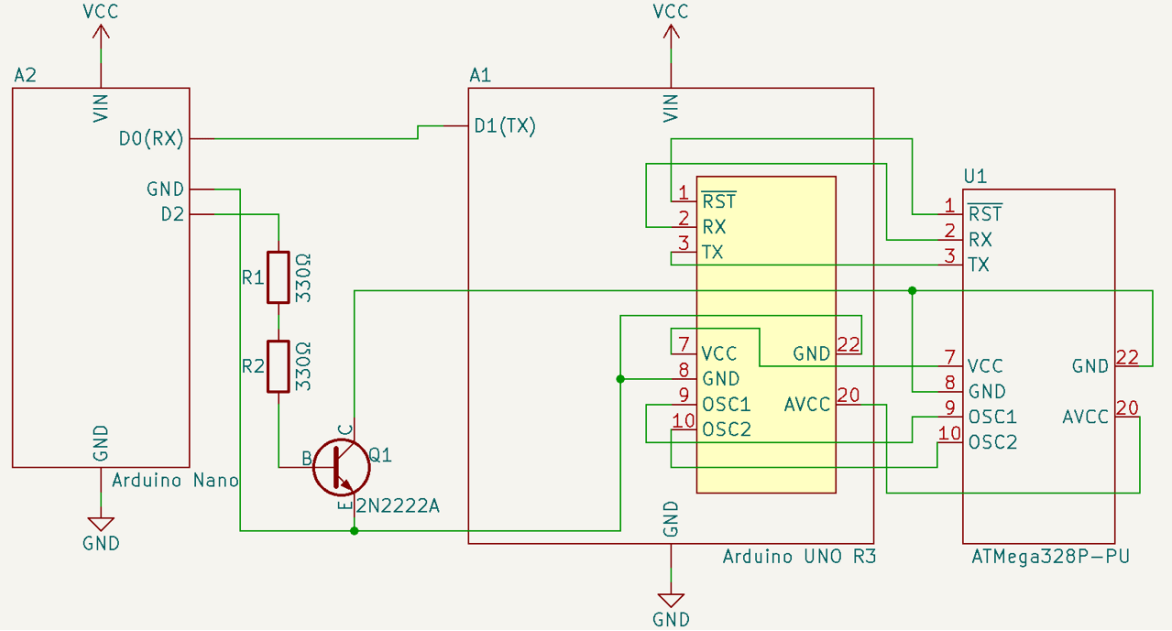
3.1.1 Postup útoku

Postup kopíruje popis pôvodného útoku [6]. Ako prvé potrebujeme nahrať firmvér, na ktorý budeme útočiť na mikrokontrolér ATMega328P, ktorý je súčasťou dosky Arduino UNO. Doska obsahuje prevodník z USB na UART, čo zjednodušuje celý postup. Prevodník na doske pripojíme k počítaču a nahráme firmvér zo súťaže pomocou softvéru AVRDUDE [3]. Uvádzame aj použitý príkaz pre nahratie firmvéru zo súboru „fiesta.hex“ [6] (súbor sa nachádza aj v elektronickej prílohe):

```
avrdude -c arduino -p atmega328p -p /dev/<zariadenie> -b115200 -u -V -U flash:w:fiesta.hex
```

Ďalším krokom je zapojenie hardvéru. Mikrokontrolér ATMega328P vyberieme z dosky Arduino UNO, keďže doska obsahuje stabilizátor napätia, ktorý by útok komplikoval a zavádzal ďalšie premenné do experimentu. Následne zapojíme tranzistor tak, aby ho bolo možné spínať pomocou Arduino Nano – emitor prepojíme so zemou na doske a bázu prepojíme s pinom, ktorý ovláda tranzistor (v našom prípade pin D2). Medzi doskami prepojíme piny zabezpečujúce sériovú komunikáciu UART smerom z UNO do Nano, aby bolo možné prečítať vypísaný „flag“, pre tento účel je potrebné prepojiť aj zem medzi doskami. Vybratý čip ATMega328P v puzdre THT umiestnime na kontaktné pole a prepojíme (káblikmi) spätne s pôvodnou doskou Arduino UNO nasledovné piny: 1, 2, 3, 7, 9, 10 a 20. Tieto zabezpečujú základné potreby pre fungovanie mikrokontroléra – napájanie (zem týmto spôsobom neprepájame), hodiny (externý oscilátor), sériová komunikácia. Piny 8 a 22 (zem), prepojíme s kolektorom tranzistora. Ovládaním tranzistora pomocou výstupného pinu D2 na Arduino Nano, potom vieme odpájať a spájať ATMega328P so zemou. Podrobná schéma celého zapojenia sa nachádza na obrázku 3.1.

Ďalej je potrebné naprogramovať Arduino Nano, aby pomocou ovládania tranzistora na krátko odpojilo napájanie ATMega328P od zeme a následne opäť pripojilo.



Obr. 3.1: Schéma zapojenia pre útok na firmvér zo súťaže CTF [6]. Žltý obdĺžnik znázorňuje puzdro na doske Arduino UNO, z ktorého bol vybratý mikrokontrolér ATMega328P.

Algoritmus 3.1: Ovládanie tranzistora, ktorý spína napájanie na ATMega328P. Prevzaté zo zdrojového kódu pôvodného útoku [6]. Cieľom premennej „waste“ je zabrániť, aby optimalizácia kompilátora odstránila for-cyklus.

```
int waste = 0;
digitalWrite(powerPin, LOW);
for (int i=0; i<glitchDelay; i++){ waste++; }
digitalWrite(powerPin, HIGH);
glitchDelay += 10;
```

Dĺžka časového intervalu, počas ktorého je tranzistor zatvorený, musí byť dostatočne dlhá, aby indukovala chybu na procesore, ale nie príliš dlhá, aby sa mikrokontrolér nereštartoval. Vyskúšali sme do Arduino Nano nahrať program použitý aj v pôvodnom útoku [6], ktorý bol implementovaný v prostredí Arduino IDE. Algoritmus zatvorí tranzistor a vykoná niekoľko prázdnych iterácií for-cyklu, po ktorom tranzistor opäť otvorí. Následne sa pokúsi prečítať výstup zo sériového portu na ATMega328P. V prípade, že sa podarí prečítať „flag“, útok bol úspešný. V prípade, že sa „flag“ nepodarí prečítať, zväčší počet iterácií for-cyklu a postup zopakuje [6]. Ukážka časti kódu, ktorá ovláda tranzistor, je v algoritme 3.1.

3.1.2 Výsledok, analýza a vylepšenie útoku

Útok bol vyskúšaný na štyri čipy ATMega328P. Na dvoch z nich (z rovnakej série výroby) sa útok úspešne podaril s podobným výsledkom ako v pôvodnom útoku [6]. „Flag“ sa dokonca podarilo prečítať už pri nulovom počte iterácií for-cyklu, postačoval najkratší možný výpadok – zatvorenie a okamžité otvorenie tranzistora pomocou funkcie „digitalWrite“ z knižnice prostredia Arduino IDE. Na druhých dvoch exemplároch (z inej série výroby) útok nebol úspešný a aj pri tom najkratšom možnom výpadku sa mikrokontrolér reštartoval. Rozhodli sme sa preto napätie na mikrokontroléri počas útoku podrobne analyzovať pomocou osciloskopu MDO4104C, ktorý sme stručne uviedli v sekcii 2.3.1. Pre tento účel sme sa rozhodli ATMega328P zapojiť na kontaktom bez-spájkovom poli bez dosky Arduino UNO. Schému a detaily tohto zapojenia sme popísali v kapitole 2 a na obrázku 2.1. Takéto zapojenie umožňuje jednoducho meniť zapojené elektronické súčiastky, čo poskytuje väčšiu flexibilitu vo voľbe parametrov týchto súčiastok pri analýze.

Pomocou osciloskopu sa podarilo určiť priebeh zmeny napätia na mikrokontroléri v čase s presnosťou na rádovo stovky nanosekúnd. Zistili sme, že interval, počas ktorého nastalo podpätie bol príliš dlhý (približne 2 µs), čo pri útoku na dva čipy zo štyroch spôsobilo reštart mikrokontroléra. Ďalším zaujímavým pozorovaním bol fakt, že časový interval podpätia sa nepredlžoval so zväčšovaním počtu iterácií „prázdneho“ for-cyklu. Dôvodom bola optimalizácia kompilátora, ktorý sa oprávnene rozhodol zdanlivo „zbytočný“ for-cyklus odstrániť, napriek inkrementu premennej „waste“ v tele for-cyklu.

Doska Arduino Nano, ktorá ovládala tranzistor obsahuje tiež mikrokontrolér ATMega328P, s externým oscilátorom s frekvenciou 16 MHz. V kapitole 2 (v sekcii 2.1) sme spomenuli ukážku nastavenia výstupnej logickej hodnoty na 1, resp. 0 pomocou jedinej inštrukcie SBI, resp. CBI. Tieto inštrukcie dokáže procesor vykonať počas dvoch taktov vďaka dvojfázovej pipeline (načítanie a vykonanie inštrukcie) [1]. Pri frekvencii 16 MHz to znamená, že zatvorenie a opätovné otvorenie tranzistora by teoreticky malo trvať 1/4 µs (perióda jedného taktu krát vykonanie inštrukcií SBI a CBI – 1/16 µs × 4 takty).

Rozhodli sme sa preto časti kódu, ktoré ovládajú tranzistor prepísať do jazyka asembler s využitím C Inline Assembly [8], ktorý je podporovaný aj kompilátorom prostredia Arduino IDE. Volania funkcie „digitalWrite“ sme teda nahradili ekvivalentnou konštrukciou pomocou inštrukcií SBI a CBI. Následne sme útok zopakovali s takto upraveným programom nahratým na Arduino Nano. Výsledkom bolo, že podpätie na mikrokontroléri trvalo približne 1/2 µs, čo je štyrikrát menej ako predtým. Dlhší čas v porovnaní s teoretickým časom (1/4 µs), bol pravdepodobne spôsobený vplyvom fyzikálnych faktorov, ktoré sme v teoretickom modeli zanedbali. Pri takto krátkom čase

Algoritmus 3.2: Procedúra oneskorenia v jazyku asembler. %0 označuje vstupný parameter – hodnota v registri.

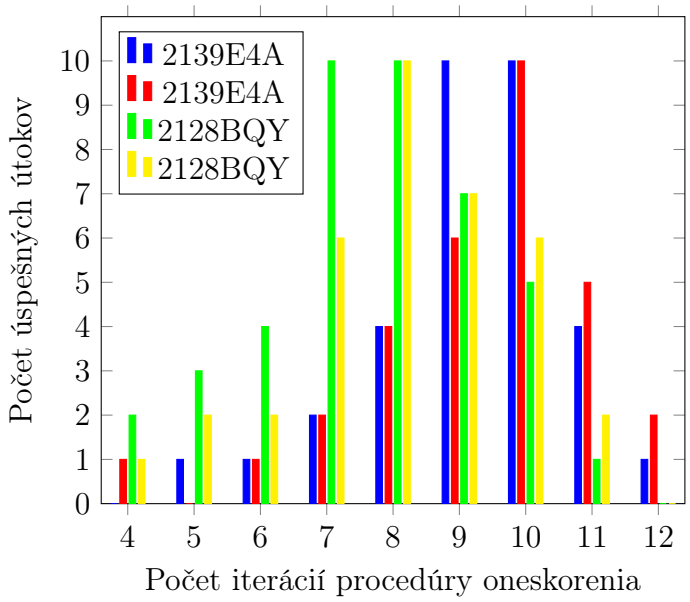
```
mov r24, %0     ; 1 takt
loop:
    dec r24       ; 1 takt
    brne loop     ; 2 takty pri vykonaní skoku, inak 1 takt
```

už nenastal reštart žiadneho z testovaných čipov, ale útok bol opäť neúspešný. Interval bol pravdepodobne príliš krátky a na žiadnom z čipov sa neprejavila chyba. Ďalej sme preto upravili kód napísaním vlastnej procedúry oneskorenia v jazyku asembler (opäť s využitím C Inline Assembly [8]). Procedúra pozostáva z inicializácie registra na kladnú hodnotu (8-bitový parameter) a cyklu. V cykle postupne dekrementujeme tento register a následne vykonáme podmienený skok na začiatok cyklu, pokiaľ výsledok dekrementovania bol nenulový. Pseudokód procedúry v jazyku asembler uvádzame v algoritme 3.2. Takáto procedúra umožňuje parametrizovať oneskorenie s presnosťou na trojice taktov (cyklus procedúry trvá tri takty). Po tejto úprave sa útok úspešne podaril na všetkých štyroch testovaných čipov. Útok sme zopakovali na každom z čipov pri rôznom počte iterácií procedúry oneskorenia (1 – 15), celkovo desaťkrát pre každú konfiguráciu (čip a počet iterácií procedúry oneskorenia). Výsledky sú znázornené na obrázku 3.2. Na čipy zo série 2128BQY pôvodný útok nebol úspešný. Možno pozorovať, že aj čipy z rovnakej série majú rôzne výsledky.

3.2 Analýza zapojenia s tranzistorom

Rozhodli sme sa ďalej podrobnejšie analyzovať priebeh napätia na mikrokontroléri medzi zatvorením a otvorením tranzistora v predošlom útoku zo sekcie 3.1. Pre tento účel sme napísali jednoduchý program, ktorý periodicky zapína a vypína tranzistor. Dĺžka časového intervalu medzi zatvorením a otvorením tranzistora je nastaviteľná pomocou analógového vstupu z potenciometra. Dĺžku tohto intervalu budeme v tejto časti udávať v počtoch iterácií procedúry oneskorenia z algoritmu 3.2, ďalej len počet iterácií oneskorenia. Tento program sme nahrali a spustili na Arduino Nano a zapojili sme ho spolu s mikrokontrolérom ATMega328P rovnako ako vo vylepšenej verzii útoku (sekcia 3.1.2). Na vstupný analógový pin dosky Arduino Nano sme zároveň pripojili potenciometer, aby ním bolo možné regulovať počet iterácií oneskorenia. Následne sme pomocou osciloskopu MDO4104C merali priebeh napätia na mikrokontroléri, pri rôznej dĺžke intervalu medzi zatvorením a otvorením tranzistora.

Ako sme očakávali, na mikrokontroléri periodicky vznikalo podpätie. Napätie počas zatvoreného tranzistora však neklesalo prudko, ale pomaly konvergovalo k hodnote

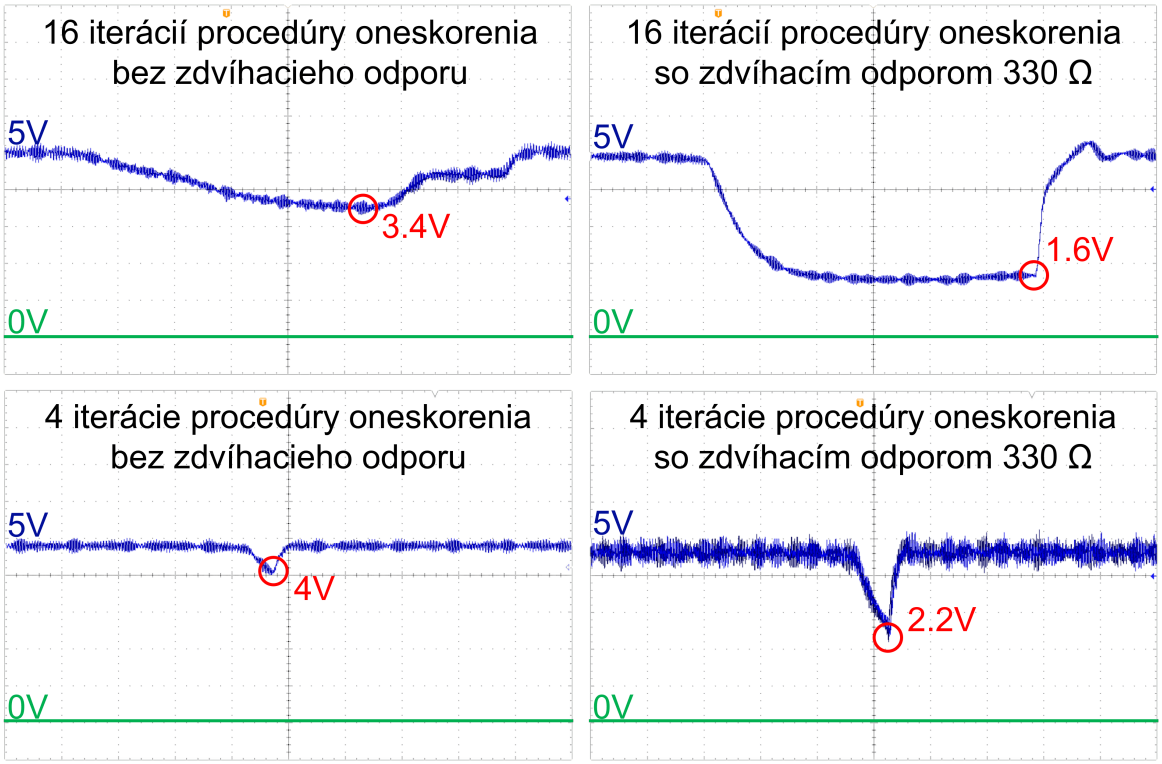


Obr. 3.2: Výsledky vylepšeného útoku zo súťaže CTF. Histogram znázorňuje počet úspešných útokov na jednotlivé čipy ATMega328P v závislosti od počtu iterácií procedúry oneskorenia z algoritmu 3.2. Každý čip je znázornený rôznou farbou, v legende je uvedená séria príslušného čipu. Znázornené sú iba konfigurácie s nenulovým počtom úspešných útokov.

približne 2 V. Po otvorení tranzistora sa napätie vrátilo (za čas približne 300 – 500 nanosekúnd) na pôvodnú úroveň 5 V. Dôsledkom takéhoto priebehu bol fakt, že pri kratších intervaloch medzi zatvorením a otvorením tranzistora napätie nestihlo klesnúť ani pod hladinu 4 V, čo už priebeh programu na cieľovom ATMega328P neovplyvnilo.

Fakt, že napätie nekleslo na 0 V, ale pokles sa zastavil pri úrovni približne 2 V, bol spôsobený tým, že k mikrokontroléru boli pripojené aj ďalšie súčiastky (najmä LED a prevodník z USB na UART). Tieto mohli mikrokontrolér nepriamo čiastočne spojiť so zvyškom obvodu aj počas zatvorenia tranzistora. Tento predpoklad sa potvrdil tým, že po odstránení týchto súčiastok a opätovnom zmeraní priebehu pomocou osciloskopu napätie už klesalo na 0 V (stále však rovnako pomaly). V ďalších častiach sa však ukáže, že fakt, že napätie neklesne až na 0 V, nebude prekážať útokom, ktoré sa pokúsime implementovať. Neskôr dokonca ukážeme, že zmena napätia až na úroveň 0 V môže v niektorých prípadoch prekážať úspešnej realizácií útoku.

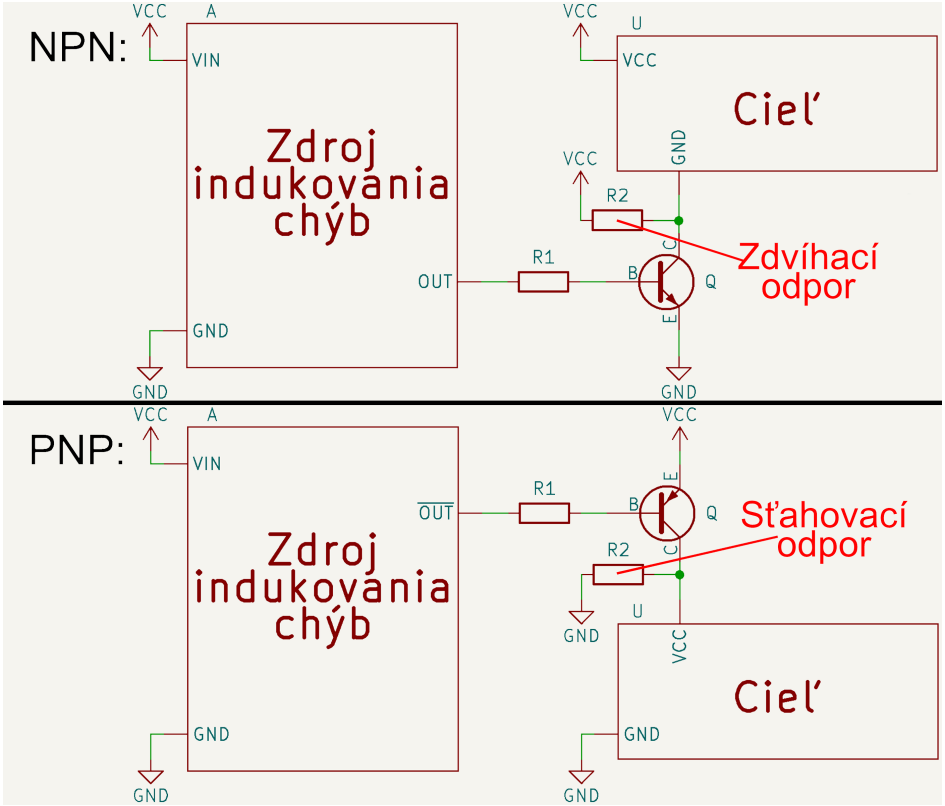
Rozhodli sme sa preto ku GND pinu mikrokontroléra pripojiť zdvíhací odpor. Tento odpor, by po zatvorení tranzistora mal podvihnúť napätie medzi zemou a pinom GND, čím zmenší napätie medzi napájacími pinmi mikrokontroléra postupne až k nule (VCC aj GND sa vyrovnajú na 5 V). Vyskúšali sme rôzne hodnoty zdvíhacieho odporu, pričom sme pozorovali, že efekt zdvíhacieho rezistora sa zväčšoval so zmenšujúcou sa hodnotou jeho odporu. Čím menší bol odpor, tým väčšie podpätie sa počas krátkeho výpadku



Obr. 3.3: Porovnanie priebehu napätia na mikrokontroléri s/bez zdvíhacieho odporu počas zatvorenia tranzistora. Veľkosť horizontálneho dielika je 400 ns. Použitý tranzistor je 2N2222A. Modrou farbou je znázornená hodnota napätia v čase a zelená čiara v spodnej časti snímok označuje úroveň napätia 0 V. Červenou je zvýraznená minimálna nameraná hodnota napätia (zaokrúhlená na desatiny voltov).

podarilo dosiahnuť. Až pri hodnote 330 Ω sme pozorovali značné zrýchlenie poklesu, čo je pomerne nízka hodnota na zdvíhací odpor. Nevýhodou je potom fakt, že pri otvorenom tranzistore rezistorom preteká pomerne veľký prúd (5 V/330 Ω = 15 mA), čo sa prejavilo citeľným zahriatím odporu počas experimentu. Na obrázku 3.3 je porovnaný priebeh napätia na mikrokontroléri s a bez použitia zdvíhacieho odporu zaznamenaný pomocou osciloskopu. Priebeh napätia sme porovnávali pri rôznom počte iterácií procedúry oneskorenia z algoritmu 3.2.

Ďalej mohli byť príčinou takéhoto priebehu aj parametre tranzistora, ale aj niektoré pasívne súčiastky v zapojení. Rozhodli sme sa preto pozorovať ako použitie rôznych tranzistorov ovplyvňuje priebeh napätia na mikrokontroléri medzi zatvorením a otvorením tranzistora. Vyskúšali sme tri rôzne bipolárne tranzistory – dva typu NPN a jeden PNP. Aby bolo možné použiť tranzistor typu PNP, bolo potrebné mierne modifikovať zapojenie. Zmena spočíva v tom, že tranzistorom budeme odpájať napájací pin, miesto zeme, čo zároveň znamená, že miesto zdvíhacieho odporu použijeme sťahovací (angl. pull-down) odpor, s rovnakým cieľom. Okrem toho vstup do bázy, ktorý riadi tranzistor bude obrátený (logická nula otvorí tranzistor). Na obrázku 3.4 je znázornený



Obr. 3.4: Všeobecná schéma zapojenia s tranzistorom typu NPN a PNP.

rozdiel medzi zapojením s tranzistorom typu NPN a PNP.

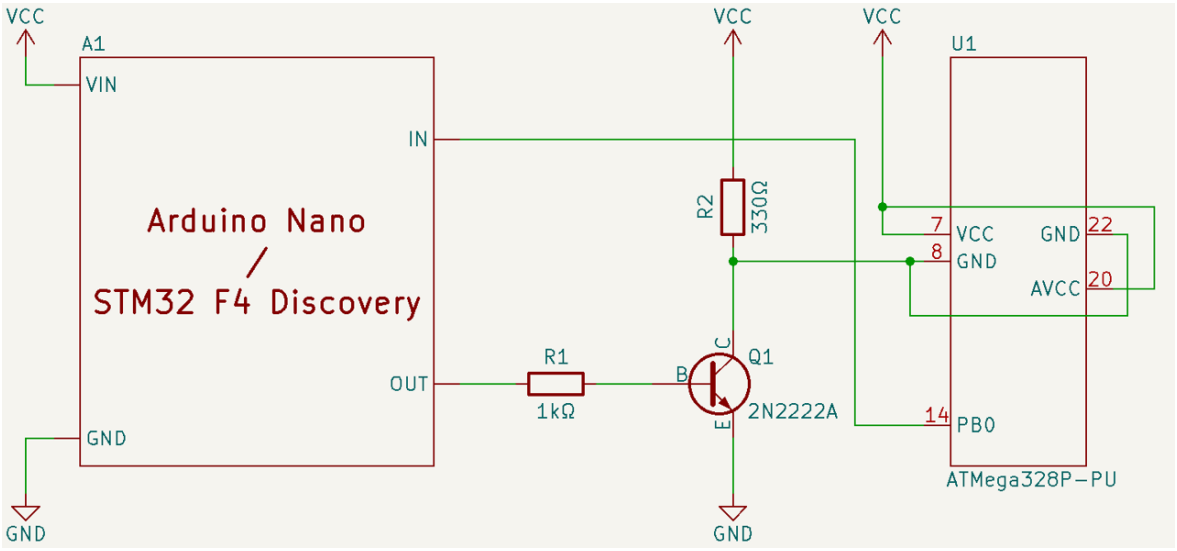
Zároveň sme každý tranzistor vyskúšali s rôznymi hodnotami zdvíhacieho, resp. sťahovacieho odporu. Výsledky meraní sú zhrnuté v tabuľke 3.1, pričom „víťazom“, ktorého sme sa rozhodli ponechať v zapojení pri ďalších útokoch bol tranzistor 2N2222A (NPN) so zdvíhacím odporom 330 Ω Tranzistor SS8050 mal síce kratšie časy stúpania a klesania napätia, ale minimálna úroveň napätia bola pri tejto konfigurácií nižšia. Dôvodom zvolenia veľmi nízkeho zdvíhacieho odporu 330 Ω bol fakt, že pri väčšom odpore bol rozdiel oproti zapojeniu bez zdvíhacieho odporu relatívne zanedbateľný. Bázový odpor tranzistora sme sa rozhodli zväčšiť na 1 kΩ, keďže dochádzalo k pomerne veľkému zahrievaniu tranzistora a prúd, ktorý potečie bude stále dostatočný. Ďalej sme overili, že táto výmena neovplyvnila priebeh napätia počas útoku.

3.3 Testovanie efektov útoku

V tejto časti podrobnejšie preskúmame, čo presne sa dá útokom využívajúcim zapojenie analyzované v predošlej sekcii 3.2. Pokúsime sa experimentálne určiť niektoré z možných efektov útoku na mikrokontroléri ATMega328P na úrovni makroinštrukcií procesora. Pomocou techniky zmeny napätia sa pokúsime ovplyvniť rôzne inštrukcie, prípadne kratšie časti kódu písané v jazyku asembler. Aby sme dosiahli väčšiu pres-

Tabuľka 3.1: Porovnanie priebehu napätia na mikrokontroléri pri zapojení s rôznymi tranzistormi. Hodnoty v tabuľke sú priemerom zo šesťdesiatich štyroch vzoriek – automaticky nameraných a vypočítaných pomocou funkcie osciloskopu. Napätie je zaokrúhlené na desatiny voltov a časy sú zaokrúhlené na stovky nanosekúnd.

Tranzistor 2N2222A (NPN)					
zdvíhací odpor (Ω)	žiaden	10k	1k	330	220
min úroveň napätia (V)	3,4	3,3	2,6	1,6	1,5
čas od zatvorenia po pokles na úroveň 3 V (ns)	N/A	N/A	1000	300	200
čas od zatvorenia po pokles na min úroveň (ns)	2000	1800	1400	1000	900
čas od otvorenia po návrat na úroveň 5 V (ns)	500	400	300	300	300
Tranzistor SS8050 (NPN)					
zdvíhací odpor (Ω)	žiaden	10k	1k	330	220
min úroveň napätia (V)	2,6	2,5	2,2	1,8	1,5
čas od zatvorenia po pokles na úroveň 3 V (ns)	1000	1000	500	300	200
čas od zatvorenia po pokles na min úroveň (ns)	1800	1700	1000	900	900
čas od otvorenia po návrat na úroveň 5 V (ns)	200	200	200	200	200
Tranzistor SS8550 (PNP)					
sťahovací odpor (Ω)	žiaden	10k	1k	330	220
min úroveň napätia (V)	3	2,9	2,7	2,1	2
čas od zatvorenia po pokles na úroveň 3 V (ns)	1000	1000	700	400	300
čas od zatvorenia po pokles na min úroveň (ns)	1000	1100	1000	1200	1200
čas od otvorenia po návrat na úroveň 5 V (ns)	200	200	200	200	200



Obr. 3.5: Schéma zapojenia s tranzistorom a synchronizáciou medzi zdrojom a cieľom útoku. Zapojenie ostatných súčiastok k ATMega328P podľa obrázku 2.1 nie je znázornené kvôli prehľadnosti. Prepojenie pinov PB0 a IN umožňuje posielanie signálov z ATMega328P do zdroja indukovania chýb, čo umožňuje synchronizáciu útoku.

nosť načasovania útoku, rozhodli sme sa pridať synchronizáciu medzi zariadením, ktoré ovláda tranzistor a mikrokontrolérom, na ktorý útočíme. Tranzistor sme ovládali dvomi rôznymi vývojovými doskami s rôznou frekvenciou procesora. Ako prvú sme vyskúšali dosku Arduino Nano, ďalej len Nano, s rovnakým mikrokontrolérom (ATMega328P), na aký útočíme. V druhom riešení sme tranzistor ovládali pomocou dosky STM32 F4 Discovery, ďalej len Discovery, ktorá by mala mať väčšiu presnosť keďže má vyššiu taktovaciu frekvenciu – 168 MHz oproti 16 MHz, ktorú má Nano. Obidva prístupy sme následne porovnali.

ATMega328P zapojíme tak, ako na obrázku 2.1 na kontaktnom bez-spájkovom poli. Zem (piny GND) zapojíme cez NPN tranzistor (aj so zdvíhacím odporom), ktorého bázu budeme ovládať pomocou výstupného GPIO pinu na zdroji indukovania chýb (Nano, resp. Discovery). Ďalej pin GPIO 8 (Port B0) na ATMega328P nastavíme ako výstupný a priamo ho pripojíme na vstupný GPIO pin zdroja indukovania chýb. Pomocou neho bude ATMega328P posielať signály pre synchronizáciu útoku. Schéma zapojenia je na obrázku 3.5. Doska Discovery neumožňuje sériovú komunikáciu s počítačom priamo cez rozhranie ST-Link, ktorým je pripojená k USB. Rozhodli sme sa preto prepojiť jej pin TX s pinom RX na ďalší samostatný prevodník z USB na UART (rovnaký model ako prevodník pripojený k ATMega328P – obrázok 2.1) pre komunikáciu s počítačom na samostatnom porte.

Experiment bude potom prebiehať nasledovne: ATMega328P pošle signál zdroju indukovania chýb a následne začne vykonávať kód, na ktorý chceme útočiť (séria inštrukcií písaná v jazyku asembler). Zdroj indukovania chýb po prijatí signálu počká

krátky čas, ďalej len posun (offset), následne zatvorí tranzistor a opäť počká krátky čas, potom tranzistor opäť otvorí. Dĺžka medzi zatvorením a otvorením tranzistora, ďalej len výpadok, a posun sú nezávisle nastaviteľné parametre, ktorými vieme nastavovať presnosť útoku. Naše očakávanie je, že nastavením posunu vieme zacieliť na konkrétnu časť úseku kódu, na ktorý útočíme a nastavením výpadku môžeme meniť vplyv útoku. Cieľom tejto sady experimentov bude určiť ako vieme pomocou nastavovania týchto parametrov ovplyvniť výsledok útoku. Ďalej sa pokúsime určiť aký druh chyby vieme na ATMega328P takýmto spôsobom indukovať, napríklad vynechanie inštrukcie, ovplyvnenie výsledku inštrukcie, podmieneného skoku a pod.

Pre nastavenie parametrov posun a výpadok sme použili podobný prístup ako v algoritme 3.2. Výhodou je možnosť nastavovania týchto parametrov aj za behu programu. Pre dosku Nano sme použili presne tú istú procedúru, zatiaľ, čo pre dosku Discovery sme navrhli analogickú, keďže architektúra jej mikrokontroléra je odlišná (podrobnosti v kapitole 2, sekcii 2.2), napríklad nepodporuje inštrukciu DEC (priamy dekrement registra). Túto inštrukciu sme nahradili pomocou inštrukcie SUBS, ktorá odčíta konštantu od registra a nastaví podmienkové bity, pre následné vykonanie podmieneného skoku. Keďže STM32F407 na doske Discovery má vyššiu taktovaciu frekvenciu, procedúra oneskorenia poskytuje väčšiu presnosť pri nastavení počtu iterácií. Porovnanie teoretickej presnosti procedúr oneskorenia sa nachádza v tabuľke 3.2. Z hľadiska presnosti je relevantný aj minimálny čas potrebný pre nastavenie výstupnej hodnoty na 0 a následne na 1 (napríklad na zatvorenie a okamžité otvorenie tranzistora). Túto hodnotu sme sa priamo pokúsili odmerať pomocou osciloskopu, pričom s doskou Nano sme namerali čas približne 200 ns, zatiaľ, čo pri doske Discovery len 100 ns. Tieto časy teda predstavujú dolný odhad pre časové oneskorenie riadenia akéhokoľvek obvodu. Zároveň sme sa pokúsili určiť presnosť procedúry oneskorenia. Odmerali sme dĺžku výpadku pri rôznom počte iterácií a následne sme vypočítali priemer rozdielu pri posune o jednu iteráciu. Zaokrúhlené hodnoty sú v tabuľke 3.2. Hodnotu parametrov posun a výpadok budeme ďalej v tejto časti uvádzať v počte iterácií procedúry oneskorenia pre dosku Nano resp. Discovery (napr. výpadok 5 znamená 5 iterácií procedúry oneskorenia medzi zatvorením a otvorením tranzistora).

3.3.1 Experimenty

Na otestovanie efektu útoku sme napísali jednoduché úseky kódu v jazyku asembler, ktoré ATMega328P spustí hneď ako pošle signál zdroju indukovania chýb. Navrhli sme tri procedúry:

- Načítanie konštanty do registra (inštrukcia LDI)
  Najskôr inicializujeme hodnotu registra R na 0x00 a následne doň uložíme hodnotu 0x55 (striedajúce nuly a jednotky v binárnom zápise). Cieľom útoku je

Tabuľka 3.2: Porovnanie teoretickej presnosti procedúry oneskorenia na STM32F407 a ATMega328P. Hodnoty v tabuľke sú vypočítané na základe parametrov, ktoré uvádza príslušný výrobca [1, 2, 17, 16].

ATMega328P (Arduino Nano)	
frekvencia procesora	16 MHz
dĺžka 1 CPU cyklu (s využitím pipeline)	1/16 µs
počet CPU cyklov 1 iterácie (priemerný)	3 CPU cykly = 3/16 µs
nameraná dĺžka 1 iterácie (priemerná, zaokrúhlená)	200 ns
minimálna dĺžka (inicializácia + 1 iterácia)	3 CPU cykly = 3/16 µs*
STM32F407 (STM32 F4 Discovery)	
frekvencia procesora	168 MHz
dĺžka 1 CPU cyklu (s využitím pipeline)	1/168 µs
počet CPU cyklov 1 iterácie (priemerný)	4 CPU cykly = 1/42 µs
nameraná dĺžka 1 iterácie (priemerná, zaokrúhlená)	20 ns
minimálna dĺžka (inicializácia + 1 iterácia)	3 CPU cykly = 1/56 µs*

*pri vykonaní iba 1 iterácie nedochádza k vykonaniu skoku a následnému sušeniu pipeline, preto dĺžka v tomto prípade môže byť kratšia ako priemerná dĺžka iterácie

ovplyvniť výsledok operácie tak, aby hodnota v registri R po vykonaní tohto kódu bola rôzna od 0x55, prípadne úplné vynechanie inštrukcie (v R ostane počiatočná hodnota 0x00).

- Priamy skok na adresu (inštrukcia RJMP)
  Inicializujeme hodnotu registra R na 0x00 a následne vykonáme skok na adresu A, ktorá sa nachádza o jednu inštrukciu ďalej. Medzi inštrukciou skoku a adresou A preskočíme inštrukciu LDI, ktorá uloží do R konštantu 0xFF). Cieľom útoku je vynechať inštrukciu skoku, čo by malo mať za výsledok, že do registra R sa uloží hodnota 0xFF (inštrukcia LDI sa nepreskočí). Za normálnych okolností by v R mala ostať hodnota 0x00.

- Séria inkrementov (inštrukcia INC)
  Opäť inicializujeme hodnotu registra R na 0x00 a následne vykonáme sériu tridsaťdva inkrementov pomocou inštrukcie INC, ktorá zakaždým zväčší hodnotu v registri o jeden. Pri korektnom behu by v registri R po vykonaní experimentu mala byť teda hodnota 32 (0x20). Cieľom bude pozorovať ako bude útok (s rôznymi hodnotami posunu a výpadku) vplývať na výsledok v registri R.

Výsledkom bolo, že v prvých dvoch experimentoch bol útok neúspešný (Nano aj

Discovery) – príliš krátky výpadok nespôsobil žiadnu pozorovateľnú chybu, dlhý mal za následok úplné zlyhanie (mikrokontrolér sa síce nereštartoval, ale posielal na výstup nezmyselné správy a prestal reagovať na vstupy). Nepodarilo sa nastaviť dĺžku výpadku (binárnym vyhľadávaním) tak, aby bol výsledok útoku niekde medzi vyššie spomenutými. Zmena dĺžky posunu výsledok neovplyvnila. Pri útoku na sériu inkrementov sa podarilo dosiahnuť, že výsledná hodnota v registri R bola 31 miesto očakávanej 32, s rovnakým výsledkom pri Nano aj Discovery.

Spomenuté pozorovanie pri útoku na sériu inkrementov naznačujú, že podpätie na mikrokontroléri počas výpadku pravdepodobne spôsobilo vynechanie jednej inštrukcie INC. Problém pri útoku v druhých dvoch experimentov mohol spočívať v tom, že kód, na ktorý cielime bol príliš krátky. Oneskorenie dané reakčným časom útočiaceho mikrokontroléra a tranzistora spôsobilo, že inštrukcia, ktorú útok ovplyvnil už nebola súčasťou cieleného kódu, ale súčasťou obsluhy rozhrania UART, ktorým ATMega328P posielalo výsledok experimentu. To mohlo mať za následok, že mikrokontrolér posielal nezmyselné správy. V horšom prípade stav pamäte nebol konzistentný z pohľadu funkcií, ktoré obsluhujú toto rozhranie, čo mohlo spôsobiť ďalšie fatálne poruchy. Pri nastavení posunu nad hraničnú hodnotu (2 v prípade dosky Nano, 47 v prípade dosky Discovery) už nebol úspešný ani útok na sériu inštrukcií INC. Našou hypotézou teda je, že týmto útokom vieme docieliť vynechanie jednej inštrukcie (vyvolaním jedného podpätia), pričom musíme počítať s istým oneskorením od prijatia signálu na zaútočenie. Toto oneskorenie vieme s obmedzenou presnosťou regulovať nastavením parametra posun, minimálna dĺžka oneskorenia je však relatívne veľká.

Rozhodli sme sa preto modifikovať kód v ostatných dvoch experimentoch, ktoré boli predtým neúspešné. Pred inštrukciu LDI, resp. RJMP sme pridalo niekoľko inštrukcií NOP. Pridaním inštrukcií NOP by sme mali vykompenzovať oneskorenie útočiaceho mikrokontroléra, ktorý by už mal stihnúť zaútočiť na cielenú inštrukciu. Našim novým cieľom je teraz potvrdiť predošlú hypotézu a zistiť počet inštrukcií NOP, ktoré treba pridať, aby bol útok aj v prípade druhých dvoch experimentov úspešný.

Rovnaký útok sme teraz vykonali na takto modifikovaný kód (pomocou Nano aj Discovery), pričom útok sme spustili s nulovým posunom (Hneď po prijatí signálu). Úspešne sa podarilo dosiahnuť vynechanie cielenej inštrukcie, po pridaní vhodného počtu inštrukcií NOP pred ňu. Počet bol pritom rôzny pri útočení pomocou dosky Nano a Discovery, podrobnosti sú v tabuľke 3.3. V prípade inštrukcie LDI zostala v registri pôvodná hodnota 0x00 a v prípade RJMP bol register prepísaný na hodnotu 0xFF inštrukciou, ktorú mal skok preskočiť. Ďalej sme skúsili útok zopakovať aj s inými hodnotami pri inicializácií a prepísaní registra. Výsledok bol analogický, čím sme potvrdili, že skutočne dochádza k vynechaniu inštrukcie a nie k inému ovplyvneniu programu.

Ďalej sme sa pokúsili overiť, či zmena parametra posun vplýva na oneskorenie útoku. Útok sme zopakovali viackrát, pričom sme postupne zväčšovali posun. So zväčšujúcim

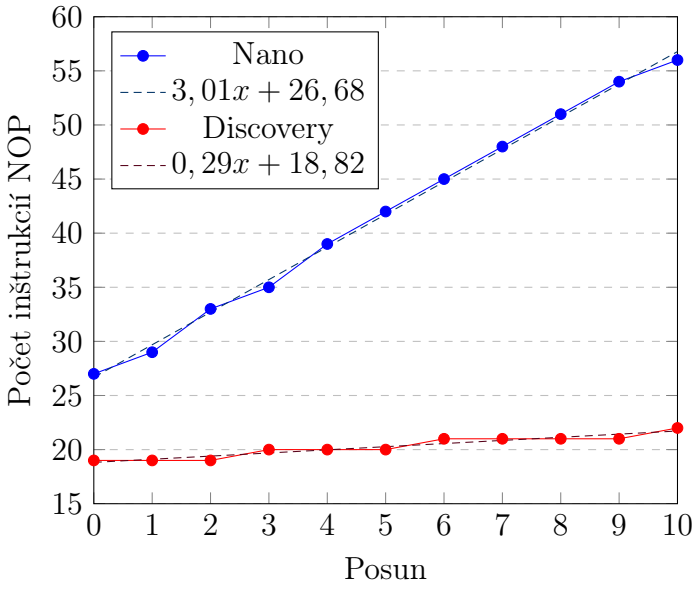
Tabuľka 3.3: Porovnanie výsledkov experimentov s doskami Nano a Discovery. Veľkosť parametra výpadok je uvedený počtom iterácií príslušnej procedúry oneskorenia. Parameter posun bol pri týchto útokoch vždy nulový.

zistený parameter	Nano	Discovery
min dĺžka výpadku (úspešný útok)	4	50
max dĺžka výpadku (úspešný útok)	7	71
ideálna dĺžka výpadku (najväčšia úspešnosť)	6	58
min počet inštrukcií NOP (úspešný útok)	26	17
ideálny počet inštrukcií NOP (najväčšia úspešnosť)	27	19
max počet inštrukcií NOP (úspešný útok)	27	20

sa posunom sa zväčšoval aj počet potrebných inštrukcií NOP, ktoré bolo treba pridať pred cielenú inštrukciu, pričom nárast bol približne lineárny. Malé odchýlky od presnej lineárnej závislosti mohli byť spôsobené tým, že zväčšenie posunu o jednu iteráciu procedúry oneskorenia časovo nezodpovedá presne celočíselnému násobku počtu inštrukcií NOP na cieľovom ATMega328P. Zistený vzťah medzi veľkosťou posunu a počtom NOP inštrukcií pred cielenou inštrukciou je znázornený v grafe na obrázku 3.6 pre hodnoty posunu 0 – 10. Pre presnejšie určenie tejto závislosti by bolo potrebné vykonať rádovo viacej testov. Našim cieľom však bolo len overiť, či zmena parametra posun ovplyvňuje útok. Pri skutočnom útoku je program, na ktorý sa útočí zvyčajne statický z hľadiska počtu inštrukcií v kóde a cieľom útočníka by skôr bolo správne nastaviť parametre posun a výpadok, čo naše riešenie automatizovane umožňuje aj dynamicky za behu.

3.3.2 Ďalšie pozorovania

Podarilo sa nám zistiť, že zapojenie s tranzistorom umožňuje cielene vynechať inštrukciu so spoľahlivou úspešnosťou. Po vhodnom nastavení parametrov posun a výpadok (v závislosti od cieleného kódu) bolo možné útok opakovať niekoľko krát za sebou (bez potreby reštartu cieľa) s úspešnosťou nad 90%. V niektorých prípadoch sa pri útoku na inštrukciu LDI dokonca podarilo dosiahnuť, že hodnota v registri nebola zhodná ani s pôvodnou, ani s hodnotou, ktorú mala cielená inštrukcia do registra prepísať. Pravdepodobne sa podarilo útok načasovať tak, že inštrukcia sa síce vykonala, ale s chybným výsledkom. Výsledná konštanta nebola zhodná ani s konštantou očakávanou pri korektnom behu programu (0x55), ani s konštantou, ktorá by mohla vzniknúť vynechaním ľubovoľnej inštrukcie (alebo viacero) z cielenej časti kódu (0x00). Takýto výsledok sa však nepodarilo spoľahlivo a stabilne reprodukovať, tak aby bolo možné cielene ovplyvniť výsledok inštrukcie. Môžeme teda konštatovať, že útok vyžadujúci takéto chybné



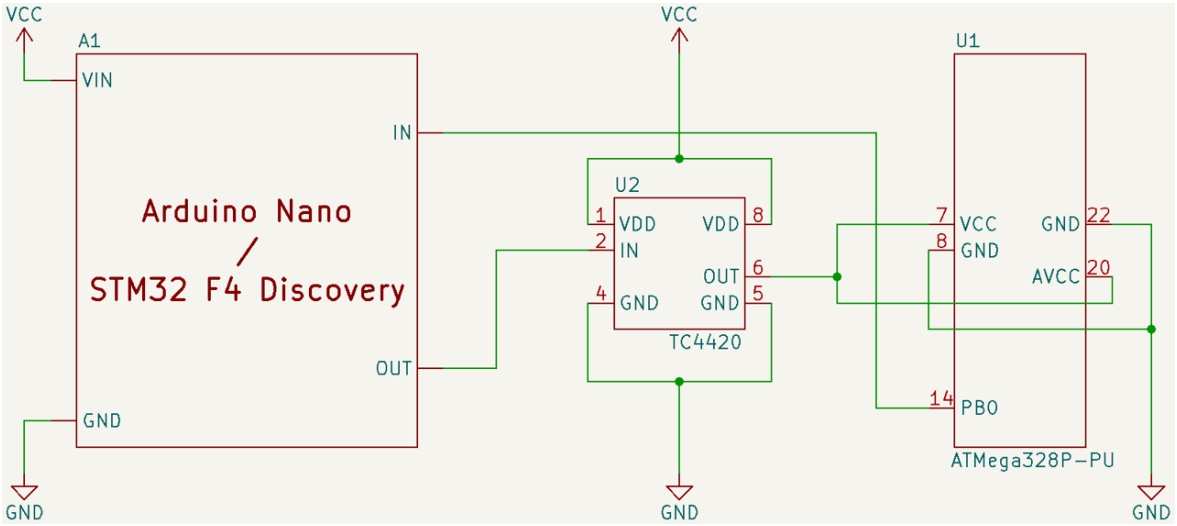
Obr. 3.6: Závislosť oneskorenia útoku od parametra posun. Na zvislej osi je počet NOP inštrukcií (určený na základe úspešnosti útoku) pred cielenou inštrukciou. Na vodorovnej osi je veľkosť parametra posun (počet iterácií príslušnej procedúry oneskorenia). Čiarkovanou čiarou je znázornená lineárna aproximácia príslušnej závislosti (zaokrúhlená na dve desatinné miesta).

správanie by pri našom zapojení vyžadoval použitie rýchlejšieho hardvéru. Rovnako sa nepreukázalo, že by sa podarilo cielene ovplyvniť inštrukciu skoku tak, aby program skočil na útočníkom zvolenú adresu. Nesprávne vyhodnotenie podmienky skoku, však možno simulovať vynechaním tejto inštrukcie v prípade, že potrebujeme, aby sa skok nevykonal.

3.4 Útok s využitím hradlového ovládača

Ďalej sa pokúsime o útok pomocou hradlového ovládača, integrovaného obvodu TC4420, ktorý sme stručne opísali v sekcii 2.3.3. Pri útoku pomocou zapojenia s tranzistorom sme pozorovali pomalý pokles napätia, ktorý sme museli vykompenzovať pridaním zdvíhacieho odporu (zdvíhame zem). Výrobca hradlového ovládača TC4420 uvádza, že časy stúpajúcich a klesajúcich hrán sú 25 ns [9], čo je dokonca rýchlejšie ako zapojenie s tranzistorom spolu so zdvíhacím odporom analyzované v sekcii 3.2.

Keďže integrovaný obvod TC4420 už obsahuje všetky potrebné súčiastky, k jeho pripojeniu k cieľovému ATMega328P nám stačia len prepojovacie káblíky a kontaktné bez-spájkové pole. Schéma zapojenia mikrokontroléra s hradlovým ovládačom je znázornená na obrázku 3.7. Pripomíname, že k mikrokontroléru ATMega328P sme pripojili aj súčiastky potrebné pre jeho fungovanie podľa schémy na obrázku 2.1. Zároveň sme ponechali prepojenie pinov z predošlého útoku, ktoré umožnilo synchronizáciu zdroja



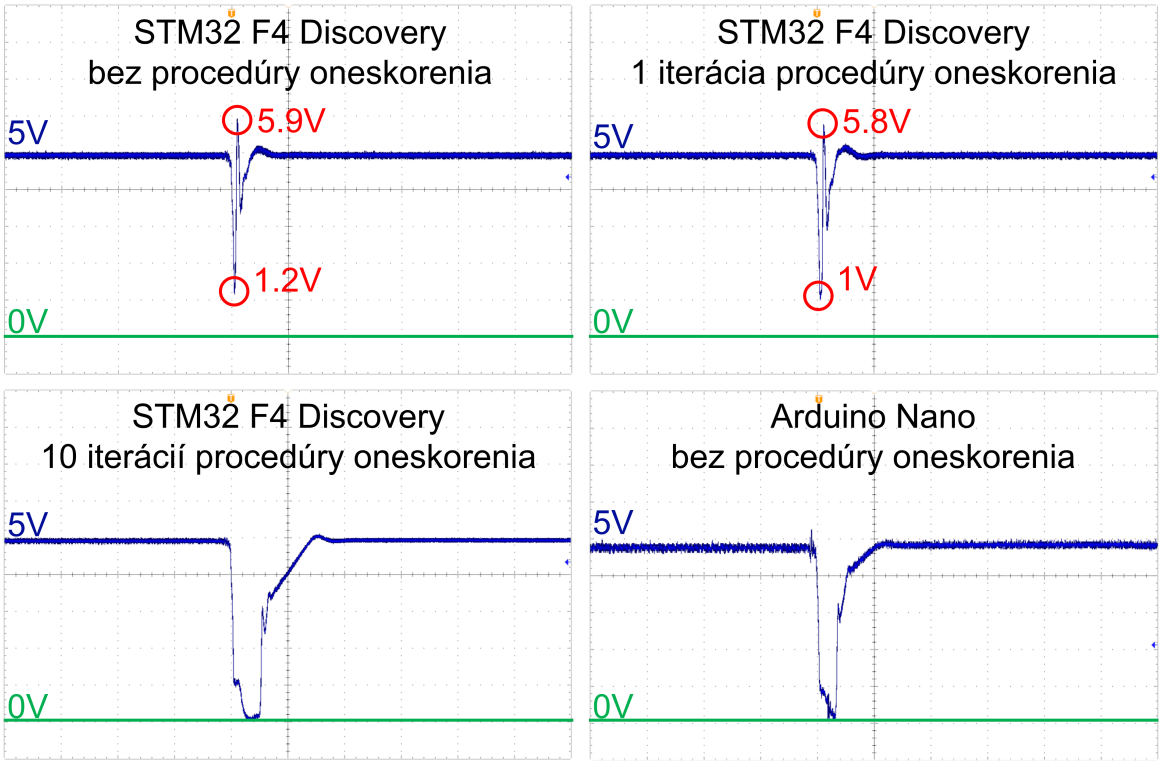
Obr. 3.7: Schéma zapojenia s hradlovým ovládačom TC4420. Okrem opísanej schémy sú k ATMega328P pripojené pomocné súčiastky podľa zapojenia z obrázku 2.1. Tie nie sú znázornené kvôli prehľadnosti.

indukovania chýb s cieľom.

Následne sme sa pokúsili útokom ovplyvniť krátky úsek kódu písaný v jazyku asembler – séria inkrementov, analogicky ako v sekcii 3.3.1. Cieľ útoku, mikrokontrolér ATMega328P, je napájaný pomocou hradlového ovládača. Tesne pred vykonaním cieleného kódu poslal signál na zahájenie útoku zdroju indukovania chýb – Doske Arduino Nano resp. STM32 F4 Discovery. Ten následne vypol a vzápätí zapol napájanie na cieli pomocou hradlového ovládača s nastaviteľným parametrom výpadok, tak ako v predchádzajúcej sekcii 3.3.1. Výsledkom bolo, že mikrokontrolér ATMega328P sa vo väčšine prípadoch reštartoval, pri použití dosky Nano ako zdroj indukovania chýb aj v najkratšom možnom výpadku – bez procedúry oneskorenia medzi vypnutím a zapnutím. V prípade dosky Discovery sa s nízkou úspešnosťou (približne 10%) útok podaril a jedna inštrukcia INC bola preskočená, podobne ako pri zapojení s tranzistorom, ale len pri jednej iterácií procedúry oneskorenia. Oneskorenie dĺžky dve iterácie už spôsobilo reštart aj v prípade dosky Discovery.

Rozhodli sme sa preto opäť analyzovať priebeh napätia na cieľovom mikrokontroléri pomocou osciloskopu MDO4104C. Zistili sme, že napätie klesalo značne rýchlejšie a až na úroveň 0 V, po zapnutí sa hodnota napätia vrátila na 5 V. Pri kratších intervaloch medzi zapnutím a vypnutím napätie stihlo klesnúť po úroveň približne 1 V a zároveň pri nábehovej hrane vznikala špička s hodnotami okolo 6 V. Na obrázku 3.8 sú znázornené snímky (zaznamenané pomocou osciloskopu), pri vybraných konfiguráciách – daných počtom iterácií procedúry oneskorenia (medzi vypnutím a zapnutím) a doskou, ktorá riadi útok (Nano, resp. Discovery).

Na základe analýzy môžeme konštatovať, že dôvodom neúspešnosti útokov pri za-



Obr. 3.8: Priebeh napätia na mikrokontroléri pri útoku pomocou hradlového ovládača. Na snímkach je znázornený priebeh napätia pri vybraných konfiguráciách. Veľkosť horizontálneho dielika je 400 ns. Modrou farbou je znázornená hodnota napätia v čase a zelená čiara v spodnej časti snímok označuje úroveň napätia 0 V. Červenou sú zvýraznené významné hodnoty napätia v čase (zaokrúhlené na desatiny voltov).

pojení s hradlovým ovládačom je pravdepodobne prudký pokles napätia až na 0 V a pomalá reakcia nami zvoleného hardvéru pre riadenie útoku. Interval, počas ktorého vzniklo podpätie bol príliš dlhý a preto väčšia kvalita zapojenia s hradlovým ovládačom v porovnaní s tranzistorom sa ukázala ako nežiadúca. Pri zapojení s tranzistorom fakt, že napätie klesalo počas vypnutia pomalšie, kompenzoval pomalosť použitého lacného hardvéru, čo v konečnom dôsledku prispelo k úspešnosti útokov. Pri použití dosky Arduino Nano sa pomocou hradlového ovládača nepodarilo realizovať ani jeden úspešný útok. Pomocou dosky STM32 F4 Discovery sa útok síce podaril, ale, ako sme spomenuli, s rádovo nižšou presnosťou ako pri zapojení s tranzistorom. Preto sme sa rozhodli týmto útok pomocou hradlového ovládača uzavrieť s tým, že k jeho úspešnému použitiu na útok pomocou techniky zmeny napätia by bol potrebný lepší hardvér s vyššou rýchlosťou, napr. FPGA alebo laboratórny zdroj. Napriek tomu sme analýzou potvrdili, že z hľadiska presnosti a reakčného času mal hradlový ovládač lepšie vlastnosti ako tranzistor.

Kapitola 4

Príklady úloh do súťaží

V tejto kapitole využijeme výsledky analýzy útokov z kapitoly 3 na prípravu ukážkových úloh do súťaží. Podarilo sa nám demonštrovať, že pomocou útoku využívajúceho techniku zmeny napätia vieme spôsobiť cielené vynechanie inštrukcie. Na útok využijeme zapojenie s tranzistorom, ktoré bolo pre naše účely dostatočne účinné. Zároveň budeme brať do úvahy obmedzenie dané nízkou rýchlosťou hardvéru riadiaceho obvod s tranzistorom. Kritickú časť kódu, na ktorú bude v príklade treba zacieliť preto implementujeme v jazyku asembler a inštrukcie zvolíme tak, aby bolo útok možné úspešne realizovať.

Konkrétne uvedieme dva príklady, jeden základný cielene implementovaný tak, aby bolo jednoduché na program zaútočiť. V druhom príklade sa budeme snažiť uviesť realistickejší program, ktorý udeľuje prístup pomocou čítačky RFID (Radio Frequency IDentification) kariet. Následne demonštrujeme úspešný útok, ktorý spôsobí, že program udelí prístup aj po prečítaní karty, ktorej prístup nemal udeliť. Súčasťou oboch príkladov bude zároveň vzorové riešenie, v ktorom popíšeme použitý program aj potrebné zapojenie pre realizáciu útoku. Pri riešení oboch príkladoch využijeme znalosť zdrojového kódu programu. (Zdrojové kódy programov z oboch príkladov sú dostupné v elektronickej prílohe priloženej k prác.) V reálnej situácii by pravdepodobne pred implementáciou útoku bolo potrebné skompilovaný kód analyzovať pomocou reverzného inžinierstva a nájsť v kóde citlivé inštrukcie, na ktoré zacieliť. Takáto analýza však presahuje rámec tejto práce a našim zámerom je demonštrovať funkčnosť samotného útoku. V elektronickej prílohe sa však nachádzajú aj skompilované HEX obrazy oboch programov, ktoré možno priamo nahrať na cieľový mikrokontrolér ATMega328P, napríklad pomocou softvéru AVRDUDE [3].

4.1 Príklad 1 – Uzamknutý čip

Ako prvý predstavíme jednoduchý príklad podobný s príkladom zo súťaže CTF, na ktorý sme útočili v sekcii 3.1. Cieľom tohto príkladu bude ukázať niektoré základné princípy postupu pri útoku pomocou indukovania chýb – identifikovať citlivú časť kódu, využitie výstupných signálov programu na synchronizáciu zdroja útoku s cieľom a správne načasovanie útoku.

Najskôr je potrebné spojazdniť program, na ktorý následne budeme útočiť. Program využíva na výstup dve signalizačné LED a sériovú komunikáciu pomocou rozhrania UART. Okrem základného zapojenia mikrokontroléra z obrázku 2.1 teda pripojíme cez 330 Ω odpor ešte dve LED – červenú k pinu GPIO 7 (Port D7) a zelenú k pinu GPIO 8 (Port B0). Schéma celého zapojenia aj s útočiacim hardvérom je na obrázku 4.1. Následne nahráme program na cieľový mikrokontrolér ATMega328P. Po spustení program po chvíli na sériový port pošle správu, že čip úspešne naštartoval. Následne začne periodicky blikať červenou LED a posielať správu „čip je v stave uzamknutý“. Zadaním tejto úlohy je teda „odomknúť“ čip.

4.1.1 Vzorové riešenie

Po analýze zdrojového kódu zistíme, že hlavný beh programu sa skladá z dvoch while-cyklov. V prvom sa mení logická hodnota na porte červenej LED (čo spôsobuje efekt blikania) a periodicky sa posiela správa „čip je v stave uzamknutý“. Pričom prvý cyklus sa opakuje kým premenná „status“ bude rovná nule. Po skončení prvého cyklu sa rozsvieti zelená LED (bit v príslušnom porte sa nastaví na logickú 1) a následne sa spustí druhý cyklus v ktorom sa periodicky posiela správa „čip je v stave odomknutý“. Ukážka popísaných častí kódu je v algoritme 4.1. Cieľom nášho útoku bude spôsobiť, aby sa prvý cyklus ukončil.

To možno vo všeobecnosti (pomocou indukovania chýb) dosiahnuť rôznymi spôsobmi, napr. ovplyvnením inštrukcie skoku alebo zápisu do premennej „status“. Nie všetky ovplyvnenia sú však jednoducho dosiahnuteľné (najmä pri použití lacnejšieho hardvéru ako sme ukázali v kapitole 3). Možno si však všimnúť, že v prvom cykle sa opakuje volanie funkcie „checkStatus“, ktorá pristupuje ku kľúčovej premennej „status“. Následne sa spustí časť kódu písaná v jazyku asembler – séria tridsiatich inštrukcií, ktoré modifikujú aj register, ktorého výsledná hodnota sa následne uloží do premennej „status“. (Previazanie častí kódu písaných v jazykoch C a asembler je zabezpečené vďaka C Inline Assembly [8].) Ukážky úryvkov kódu z funkcie „checkStatus“ sú v algoritme 4.2. Sémantika tejto časti programu vyzerá na prvý pohľad nezmyselne – nepravidelné striedanie zdanlivo náhodných inštrukcií, ktoré modifikujú registre. Po hlbšej analýze napríklad prekopírovaním inštrukcií do emulátora architektúry AVR a dyna-

Algoritmus 4.1: Ukážka kódu hlavnej časti programu z príkladu 1.

```
// 1. while-cyklus (status je globálna premenná typu uint8_t)
while(!status) {
    Serial.println("Chip status: locked"); // výpis na sériový port
    digitalWrite(RED_LED, HIGH); // zapnutie červenej LED
    checkStatus(); // táto funkcia pristupuje k premennej status
    delay(500);
    digitalWrite(RED_LED, LOW); // vypnutie červenej LED
    delay(500);
}

digitalWrite(RED_LED, LOW); // vypnutie červenej LED
digitalWrite(GREEN_LED, HIGH); // zapnutie zelenej LED

// 2. while-cyklus (cieľom je, aby sa sem program dostal)
while(1) {
    Serial.println("Chip status: unlocked"); // výpis na sér. port
    delay(1000);
}
```

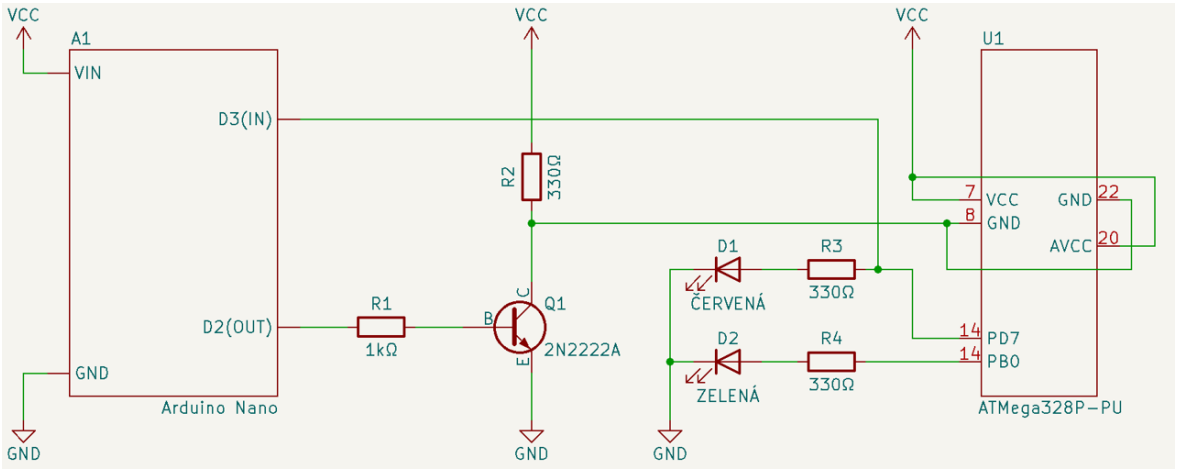
mickej analýze jeho správania zistíme, že sled inštrukcií je zvolený tak, aby výsledok bol vždy rovnaký – v registri previazaným s premennou „status“ bude uložená hodnota nula. To spôsobí, že program beží dokola v prvom cykle.

Postupnosť inštrukcií bola zámerne zvolená tak, aby zásah do (takmer) každej inštrukcie spôsobil, že výsledok už nebude rovný nule. Táto časť kódu je preto pomerne citlivá na útok pomocou indukovania chýb – vynechanie ľubovoľnej inštrukcie z tejto časti s vysokou pravdepodobnosťou spôsobí, že výsledkom už nebude hodnota nula a program prejde do druhého cyklu, čím sa čip odomkne. Aby to bolo možné dosiahnuť, je potrebné riadenie útoku správne načasovať. Pre mierne sťaženie úlohy sú navyše niektoré inštrukcie zámerne „zbytočné“ a ich vynechanie neovplyvní výsledok. Pre načasovanie útoku možno využiť samotný program bežiaci na cieli. Tesne pred zavolaním funkcie „checkStatus“ sa volá funkcia, ktorá zapína červenú LED. Tento výstup možno využiť ako signál pre synchronizáciu. Zároveň bude treba útok od prijatia signálu správne načasovať tak, aby sa trafil do kritickej časti kódu. Navyše, ako bolo spomenuté, niektoré inštrukcie nemajú vplyv na výsledok. Preto ich vynechanie nepovedie k úspešnému útoku. Napriek tomu bude načasovanie pomerne jednoduché. Necielime totiž na konkrétnu inštrukciu, ale stačí trafiť (takmer) ktorúkoľvek inštrukciu v rámci funkcie „checkStatus“.

Využijeme teda útok pomocou techniky zmeny napätia, konkrétne zapojenie s tran-

Algoritmus 4.2: Ukážky kritickej časti kódu funkcie „checkStatus“ z príkladu 1. %0 označuje výstupný parameter – register s výstupnou hodnotou. Tento parameter je previazaný s globálnou premennou status pomocou C Inline Assembly [8].

```
; inicializácia registrov
ldi %0, 0xFF
ldi r24, 0x20
ldi r25, 0x0F
ldi r26, 0xAA

; začiatok kódu, ktorý manipuluje s registrami
mov r3, r26
dec %0
add %0, r24
sub %0, r25
or %0, r26
add r26, r25
add %0, r26
clr r26          ; inštrukcia clr vynuluje register
sub r26, r24
eor %0, r3       ; inštrukcia xor v architektúre AVR
; ...

; záver
ldi r25, 0x2F
or %0, r25
inc %0 ; v registri pod týmto parametrom bude výstupná hodnota
```



Obr. 4.1: Schéma zapojenia pre útok na príklad 1 – Uzamknutý čip. K mikrokontroléru ATMega328P sú pripojené aj ostatné komponenty (nie sú znázornené kvôli prehľadnosti) podľa schémy 2.1 z kapitoly 2.

zistorom, ktorý sme analyzovali v sekcii 3.2. Ako riadiaci hardvér zvolíme dosku Arduino Nano a využijeme aj rovnaký program zo sekcie 3.3 v kapitole 3, bude však treba upraviť parameter posun. Program nastavíme tak, aby začal s hodnotou posunu jeden a postupne ju bude zväčšovať po každom pokuse o útok o jeden. V sekcii 3.3 sme pozorovali, že pri doske Arduino Nano predstavovalo zväčšenie posunu o jeden pridanie približne 200 ns. Maximálnu hodnotu parametra posun môžeme preto nastaviť napríklad na desať čo predstavuje dve mikrosekundy (10 × 200 ns). V sekcii 3.3 sme ďalej odhadli priemernú dĺžku jednej inštrukcie (na cieľovom ATMega328P) na približne 1/16 µs. Dve mikrosekundy posunu teda predstavujú približne tridsaťdva inštrukcií, čo zďaleka presahuje potrebné oneskorenie pre zásah do citlivej časti kódu. Nastavenie väčšej hodnoty posunu už nemá preto veľký zmysel. Po dosiahnutí maximálnej hodnoty parametra posun sa táto hodnota obnoví na jeden a takto bude program pokračovať až kým sa útok nepodarí. Schéma zapojenia aj s doskou Nano, ktorá riadi útok je na obrázku 4.1.

Popísaný postup sme následne implementovali a útok sa podarilo úspešne realizovať. Útok bol úspešný hneď pri hodnote jeden parametra posun. Program rozsvietil zelenú LED a začal posielať správu „čip je v stave odomknutý“. Pri hodnote parametra posun viac ako jeden bol útok už neúspešný (posun bol pravdepodobne príliš dlhý). V prípade, že by sme útok riadili pomocou dosky Discovery, postup by bol analogický. Akurát by bolo opäť potrebné správne nastaviť parameter posun. Pravdepodobne na väčšiu hodnotu, keďže doska Discovery má väčšiu taktovaciu frekvenciu procesora. Keďže na úspešný útok postačovala doska Nano, dosku Discovery sme v tomto príklade nepoužili. Riadenie útoku oboma doskami sme pre naše účely dostatočne porovnali pri analýze v kapitole 3.

4.2 Príklad 2 – Čítačka RFID kariet

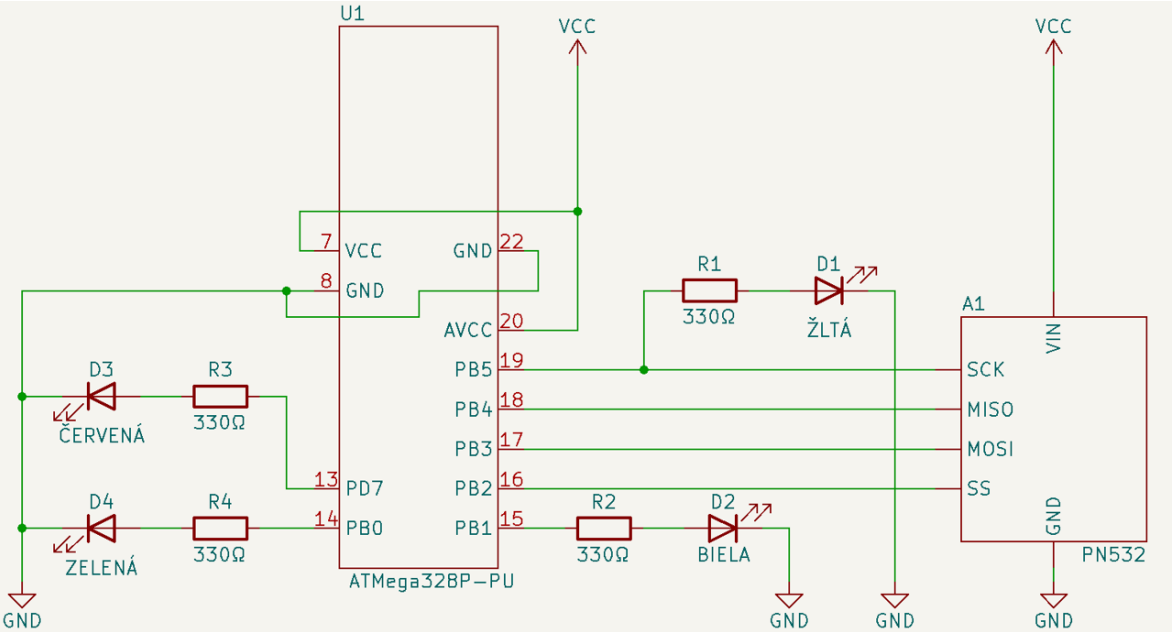
Druhým príkladom je program, ktorý udeľuje prístup na základe čítačky RFID (Radio Frequency Identification) kariet, ďalej len čipová karta. Cieľom tohto príkladu je demonštrácia útoku na realistickejší program v porovnaní s prvým príkladom. Inšpiráciu programu na čítanie čipových kariet sme prevzali z ročníkového projektu Deadlock [19]. Program sme pre naše účely do značnej mieri zjednodušili. Pôvodný program predstavuje čítačku kariet, ktorá zároveň komunikuje so serverom a na základe tejto komunikácie sa rozhoduje, či udelí prístup. Z programu sme odstránili komunikáciu so serverom a rozhodnutie o udelení prístupu sme „zadrôtovali“ do programu porovnaním s konštantou. Toto porovnanie sme implementovali v jazyku asembler spôsobom, aby bolo jednoduchšie na program zaútočiť. Túto časť kódu bližšie analyzujeme vo vzorovom riešení.

Zároveň bolo potrebné modifikovať zapojenie hardvéru, keďže pôvodný program nebol implementovaný pre mikrokontrolér ATMega328P. Pre zapojenie hardvéru budú potrebné nasledovné komponenty:

- ATMega328P-PU – THT puzdro
- kontaktné bez-spájkové pole
- modul PN532 pre čítanie RFID kariet pomocou NFC (Near Field Communication) – zapojíme cez rozhranie SPI (Serial Peripheral Interface)
- rezistory – 4-krát 330 Ω k LED
- 4 kusy LED – rôzne farby (červená, zelená, žltá, biela)
- prepojovacie káblíky typu M-M
- všetky ostatné súčiastky použité v základnom zapojení mikrokontroléra ATMega328P zo sekcie 2.1.3.

Najskôr zapojíme základné komponenty k mikrokontroléru podľa zapojenia z obrázku 2.1 z kapitoly 2. Ostatné časti doplníme podľa zapojenia na obrázku 4.2. Toto zapojenie ešte neobsahuje hardvér potrebný pre útok, ale predstavuje korektné zapojenie pre fungovanie programu.

Teraz môžeme nahrať a spustiť program na cieľovom ATMega328P. Po spustení program, pošle správu na sériový port, že bol úspešne naštartovaný. V prípade, že sa nepodarí nadviazať komunikáciu s modulom PN532 pre čítanie RFID kariet, program pošle chybovú hlášku. Pokiaľ nastane tento problém, pravdepodobne ide o nesprávne zapojenie hardvéru. Pokiaľ všetko prebehne správne, spustí sa hlavný cyklus, v ktorom sa program pokúsi prečítať čipovú kartu podľa štandardu ISO14443A. Pokiaľ čítačka



Obr. 4.2: Schéma zapojenia mikrokontroléra v príklade 2 – Čítačka RFID kariet. Okrem znázorneného zapojenia sú k mikrokontroléru pripojené aj základné súčiastky z obrázku 2.1. (Zapojenie žltej LED sme pre jednoznačnosť uviedli aj v tomto obrázku.)

žiadnu čipovú kartu nedeteguje, pošle na sériový port správu, že čaká na priloženie karty. Tento proces sa periodicky opakuje. Po priložení karty a úspešnom prečítaní, sa na krátku chvíľu rozsvieti biela LED na indikáciu, že čítanie bolo úspešné. Následne sa rozsvieti červená alebo zelená LED, ktorá indikuje zamietnutie, resp. povolenie prístupu, zároveň program pošle príslušnú správu na sériový port. Program je nastavený tak, aby udelil prístup jedinej čipovej karte a to karte s konkrétnym UID (Unique Identifier) rovným 0x348BD1A3 (v poradí od najvýznamnejšieho bajtu). Zadaním tejto úlohy je pomocou indukovania chýb prinútiť mikrokontrolér, aby udelil prístup (indikované rozsvietením zelenej LED) po prečítaní karty s nesprávnym UID, ktorej prístup nemal udeliť.

4.2.1 Vzorové riešenie

Opäť bude prvým krokom analýza zdrojového kódu. Keďže zdrojový kód príkladu 2 je mierne rozsiahlejší, pre podrobnosti ohľadom popisovaných častí kódu odkazujeme čitateľa do elektronickej prílohy, v ktorej sa nachádza kompletný (príslušne okomentovaný) zdrojový kód (cesta k programu: CTFexamples/CTF-RFID/). Podstatné časti kódu však popíšeme aj v práci.

Rozhodnutie o udelení prístupu program urobí na základe návratovej hodnoty (typu bool) funkcie „checkAccess“. Hlavná časť funkcie „checkAccess“ je napísaná v jazyku asembler (s využitím C Inline Assembly [8]). Ako prvé funkcia nastaví návratovú hod-

notu na nulu (prístup zamietnutý). Zároveň vynuluje register R0. Následne sa spustí séria porovnaní, bajt po bajte, medzi UID prečítaným z priloženej čipovej karty (v registroch) a staticky definovaným UID karty, ktorej má byť prístup udelený. Jedno porovnanie pozostáva zo samotného porovnania registra s konštantou a podmieneného skoku, ktorý preskočí práve jednu inštrukciu v prípade rovnosti. Inštrukcia, ktorú skok preskočí je inkrement registra R0, ktorý sa tým pádom vykoná iba v prípade, že porovnanie skončilo nenulovým výsledkom. Ukážka tejto časti kódu v jazyku asembler je v algoritme 4.3. Týmto spôsobom sa porovná najskôr dĺžka prečítaného UID (podľa štandardu ISO14443A môže UID mať dĺžku štyri alebo sedem bajtov) a následne najmenej významné štyri bajty prečítaného UID.

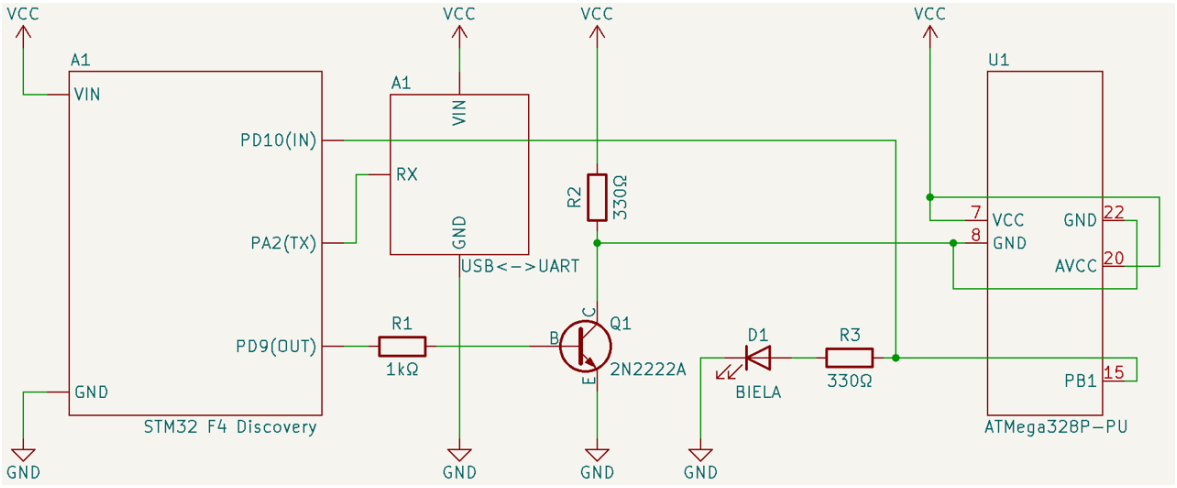
Pokiaľ všetky porovnania skončili nulovým výsledkom (rovnosť) v registri R0 zostane inicializovaná hodnota nula, v opačnom prípade bude R0 obsahovať nenulovú hodnotu (aspoň jeden inkrement sa vykoná). Na záver sa vykoná test na nulu registra R0, pokiaľ bola hodnota nulová do registra obsahujúceho návratovú hodnotu funkcie sa zapíše hodnota jeden (čo znamená „prístup povolený“). V prípade, že by priložená karta mala UID dĺžky inej ako štyri bajty, nebudú sa rovnať dĺžky, teda prvé porovnanie skončí nenulovou hodnotou a prístup nebude udelený. Dôvodom prečo algoritmus porovnania neskončí hneď pri prvej nerovnosti je znemožnenie útoku postranným kanálom, konkrétne časovým útokom (angl. timing attack). Popísaná implementácia má tú vlastnosť, že porovnanie trvá rovnako dlho nezávisle od počtu zhodujúcich sa bajtov.

Nevýhodou však je, že výsledok celého porovnania závisí na jednom podmienenom skoku (po záverečnom teste registra R0). Pokiaľ sa tento skok nevykoná návratová hodnota z funkcie bude jeden a prístup bude udelený. Táto inštrukcia je potom vhodný cieľ nášho útoku. Zároveň hneď pred zavolaním funkcie „checkAccess“, na ktorú chceme útok zacieliť, je v programe volanie funkcie „digitalWrite“, ktorá rozsvieti bielu LED. Pomocou tohto signálu môžeme, rovnako ako v príklade 1, synchronizovať útok s cieľovým mikrokontrolérom.

Opäť budeme na program útočiť pomocou zapojenia s tranzistorom, z kapitoly 3. Keďže budeme cieliť na konkrétnu inštrukciu, tentokrát bude útok riadený pomocou dosky Discovery, o ktorej sme ukázali, že poskytuje väčšiu presnosť ako doska Nano. Zapojenie útočiaceho hardvéru je podobné ako v príklade 1, jeho schéma je na obrázku 4.3. Pre posielanie správ o nastavených parametrov útoku z dosky Discovery do počítača sme pripojili ďalší prevodník z USB na UART (rovnaký model ako v zapojení z obrázku 2.1), podobne ako pri útoku v sekcii 3.3. Program pre riadenie útoku bude opäť rovnaký ako v sekcii 3.3 (program pre dosku Discovery). Analogicky ako v príklade 1 zo sekcie 4.1, bude potrebné nastaviť rozsah parametra posun, ktorý sa bude s každým pokusom automaticky inkrementovať. Rozsah parametra posun sme nastavili od jeden po sto. V kapitole 3 sme zistili, že doska Discovery je približne o jeden rád presnejšia ako doska Nano.

Algoritmus 4.3: Ukážky kódu v jazyku asembler z funkcie „checkAccess“ z príkladu 2. %0 označuje výstupný parameter – register pre návratovú hodnotou. Ostatné parametre (označené %) sú argumentami jednotlivých inštrukcií CPI. CPI porovná vždy register – prečítaný bajt UID a konštantu – bajt UID „správnej karty“ (staticky deklarovaný).

```
; inicializácia – vynulovanie registrov
clr %0              ; inštrukcia clr vynuluje register
eor r0, r0          ; inštrukcia xor v architektúre AVR

; porovnania prvého bajtu (dĺžka UID)
cpi %1, %2          ; porovnanie prečítanej dĺžky UID
breq skip_1         ; podmienený skok ak nastala rovnosť
inc r0              ; inkrement registra R0

; porovnania druhého bajtu (prvý bajt UID)
skip_1:             ; návestie pred ďalším porovnaním
cpi %3, %4          ; porovnanie prvého bajtu UID
breq skip_2         ; podmienený skok ak nastala rovnosť
inc r0              ; inkrement registra R0

skip_2:             ; návestie pred ďalším porovnaním
; ...

; ...
breq skip_5         ; podmienený skok po poslednom porovnaní
inc r0              ; inkrement registra R0

skip_5:             ; návestie pred záverečným testom R0
tst r0              ; test na nulu registra R0

; na túto inštrukciu cielime útok:
brne fail           ; podmienený skok ak R0 bol nenulový

ldi %0, 0x01        ; uloženie hodnoty 1 do výstupu
fail:
nop                 ; posledná inštrukcia v jazuku asembler

; nasleduje návrat z funkcie (už v jazyku C)
```



Obr. 4.3: Schéma zapojenia pre útok na príklad 2 – Čítačka RFID kariet. Opäť, kvôli prehľadnosti, nie sú znázornené súčiastky zo zapojení 2.1 a 4.2, ktoré je tiež potrebné k ATMega328P pridať. (Zapojenie bielej LED sme pre jednoznačnosť uviedli aj v tomto obrázku.)

Útok sme implementovali podľa uvedeného postupu. Pri opakovanom prikladaní čipovej karty (s UID, ktorému program nemá povoliť prístup) k čítačke sa zakaždým spustil útok riadený doskou Discovery. Útok bol synchronizovaný na základe signálu, ktorý vyslal cieľový mikrokontrolér tým, že po priložení karty rozsvietil bielou LED na upozornenie, že úspešne prečítal čipovú kartu. Po každom pokuse o útok doska Discovery automaticky zväčšila parameter posun až kým sa úspešne podarilo dosiahnuť, aby mikrokontrolér zasvietil zelenou LED a zároveň poslal správu „prístup povolený“, napriek tomu, že priložená karta nemala mať povolený prístup. (UID bolo rôzne od 0x348BD1A3 čo je jediné, ktorému program udelí prístup.) Hodnoty parametra posun, pri ktorých sa útok úspešne podaril boli 33 a 34. Pri menšej, alebo väčšej hodnote buď program prístup zamietol (bez známok poruchy), alebo zlyhal a mikrokontrolér sa reštartoval.

Následne sme sa pokúsili útok zopakovať riadením pomocou dosky Nano. Tentokrát bol však útok neúspešný pre všetky hodnoty parametra posun v rozmedzí 1 – 20. Väčšie hodnoty nemalo zmysel skúšať, keďže rádovo presahujú očakávané časove oneskorenie od prijatia signálu po vykonanie cieľovej inštrukcie.

Záver

Hlavným cieľom tejto práce bolo zistiť či aj lacným hardvérom možno úspešne zaútočiť pomocou indukovania chýb a cielene tak ovplyvniť správanie hardvéru. Najdôležitejším výsledkom bolo, že aj pomocou veľmi jednoduchého mikrokontroléra ako je ATMega328P (súčasť dosky Arduino Nano) vieme dosiahnuť vynechanie vybranej inštrukcie. Ďalej sme overili, že presnosť, ktorú nám ATMega328P poskytuje je pomerne obmedzená a útok na reálny program by bol s vysokou pravdepodobnosťou neúspešný. S využitím o čosi drahšieho (stále však veľmi lacného) prístupu, pomocou dosky STM32 F4 Discovery, sa nám podarilo o jeden rád zlepšiť presnosť útoku, čo umožnilo s väčšou úspešnosťou trafiť cielenú inštrukciu.

Pri implementácii jednotlivých útokov sme využili techniku zmeny napätia, konkrétne sme porovnali dva prístupy – zapojenie pomocou tranzistora a využitie hradlového ovládača. Zapojenie s tranzistorom, ktoré sme prevzali z útoku na firmvér v rámci súťaže CTF [6], sa nám podarilo vylepšiť pridaním zdvíhacieho odporu. Pri obidvoch zapojeniach (tranzistor aj hradlový ovládač) sme priebeh zmeny napätia analyzovali pomocou osciloskopu. Prekvapivým výsledkom bolo, že napriek teoreticky lepšiemu priebehu zmeny napätia pri zapojení s hradlovým ovládačom, malo zapojenie s tranzistorom väčšiu úspešnosť útokov. Ukázali sme teda, že v niektorých prípadoch môže zapojenie s pomalšími súčiastkami kompenzovať malú presnosť riadiaceho hardvéru. Respektíve, nie je potrebný pokles napätia až na 0 V, naopak lepší a zároveň postačujúci je pokles len na úroveň napríklad 2 V. Slabší pokles napätia stačil na vynechanie inštrukcie a zároveň nenastal reštart mikrokontroléra.

Zároveň sme upravili pôvodný zdrojový kód pre riadiaci hardvér útoku [6] prepísaním dôležitých častí kódu do jazyka asembler, čím sme značne zlepšili presnosť časovania útoku. Tým sme ukázali, že aj malé oneskorenie, spôsobené napríklad volaním funkcie, môže prekážať úspešnosti útoku. Navyše sme pozorovali, že ten istý útok (rovnaký hardvér, zapojenie, program) môže mať rôzny vplyv na exempláre toho istého mikrokontroléra z rôznej série výroby.

Výsledky analýzy útokov sme následne demonštrovali v rámci ukážky útoku na dva jednoduché programy. V obidvoch prípadoch sa nám podarilo úspešne zaútočiť na daný program s využitím znalostí zistených pri podrobnej analýze útoku využívajúcom zapojenie s tranzistorom. Naša demonštrácia aplikácie útokov využívala však programy

cielene implementované tak, aby bolo na ne možné jednoducho zaútočiť. Priamym nadviazaním na našu prácu by preto mohlo byť overenie ako použiteľné je nami použité zapojenie s tranzistorom a riadiacim hardvérom pri útoku na reálny firmvér skutočného produkčného zariadenia. Ďalšou zaujímavou otázkou je, či môže aj samotný kompilátor pomôcť odolnosťou voči útokom pomocou indukovania chýb. Napríklad vkladaním náhodných oneskorení, používaním netriviálnych konštánt (rôznych od 0x00, 0x01) a pod.

Ďalším výsledkom práce je, že sa nepodarilo dosiahnuť iný druh chyby ako vynechanie inštrukcie. Pre indukovanie iného typu chýb, napríklad ovplyvnenie výsledku inštrukcie, bola presnosť hardvéru nedostatočná, prípadne by bolo potrebné použiť zapojenie s iným princípom zmeny napätia alebo inú techniku indukovania chýb. Námetov na ďalšie práce týkajúce sa útokov pomocou indukovania chýb je viacero. Príkladom môže byť analýza iných techník spomenutých aj v tejto práci ako manipulácia hodín a elektromagnetické rušenie. Rovnako sa možno venovať aj obranným mechanizmom voči indukovaniu chýb, či už vývojom aktívnych mechanizmov pre detekciu takýchto útokov alebo návrhom bezpečného písania, dokonca aj kompilovania programov, ktoré by mohli skomplikovať, prípadne úplne znemožniť jednotlivé útoky.

Útoky na hardvér pomocou indukovania chýb predstavujú reálnu hrozbu a je potrebné pri návrhu, najmä bezpečnostných zariadení, aj s takýmto scenárom útoku počítať. Našou prácou sme poukázali na to, že útok pomocou indukovania chýb možno potenciálne realizovať aj s minimálnymi nákladmi. Pri návrhu systému je preto dôležité myslieť nie len na bezpečnosť softvéru, ale rovnako aj na bezpečnosť samotného hardvéru, ktorý bude v konečnom dôsledku daný softvér vykonávať.

Literatúra

[1] Atmel Corporation. *ATMega328P Datasheet*, 2015. Rev. 7810D–AVR–01/15.

[2] Atmel Corporation. *AVR Instruction Set Manual*, 2016. Rev. Atmel-0856L-AVR-Instruction-Set-Manual_Other-11/2016.

[3] AVRDUDE Development Team. AVRDUDE: An AVR programmer/utility, 2023. [Citované 2023-04-28] Dostupné z `http://savannah.nongnu.org/projects/avrdude/`.

[4] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 105–114, 2011.

[5] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.

[6] Christoffer Claesson. Voltage glitching on the cheap, 2019. [Citované 2023-03-18] Dostupné z `https://blog.securitybits.io/2019/06/voltage-glitching-on-the-cheap/`.

[7] Shaked Delarea and Yossi Oren. Practical, low-cost fault injection attacks on personal smart devices. *Applied Sciences*, 12(1), 2022.

[8] GNU Compiler Collection. Using assembly language with C, 2023. [Citované 2023-04-18] Dostupné z `https://gcc.gnu.org/onlinedocs/gcc/Using-Assembly-Language-with-C.html#Using-Assembly-Language-with-C`.

[9] Microchip Technology. *TC4420/TC4429 Datasheet*, 2012. Rev. D.

[10] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 77–88, 2013.

[11] NewAE Technology Inc. Tutorial A2 introduction to glitch attacks (including glitch explorer), 2019. [Citované 2023-01-31] Dostupné z `https://wiki.newae.com/V4:Tutorial_A2_Introduction_to_Glitch_Attacks_(including_Glitch_Explorer)`.

[12] Colin O'Flynn. Fault injection using crowbars on embedded systems. Cryptology ePrint Archive, Paper 2016/810, 2016.

[13] Colin O'Flynn. Building against fault injection attacks, 2020. [Citované 2022-04-11] Dostupné z `https://circuitcellar.com/research-design-hub/building-against-fault-injection-attacks/`.

[14] Colin O'Flynn and Zhizhang David Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 243–260. Springer, 2014.

[15] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 2–12. Springer Berlin Heidelberg, 2003.

[16] STMicroelectronics. *PM0214 STM32 Cortex-M4 Programming manual*, 2020. Rev. 10.

[17] STMicroelectronics. *RM0090 STM32F4 Reference manual*, 2021. Rev. 19.

[18] Jan Van den Herrewegen, David Oswald, Flavio D. Garcia, and Qais Temeiza. Fill your boots: Enhanced embedded bootloader exploits via fault injection and binary analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):56–81, 2020.

[19] Dennis Vita and Alex Bajús. Deadlock, 2022. [Citované 2023-04-16] Dostupné z `https://www.st.fmph.uniba.sk/~vita3/rp/zima/rp_ZS.html`.

[20] Loic Zussa, Jean-Max Dutertre, Jessy Clediere, and Bruno Robisson. Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter. In *Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 130–135, 2014.

Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádzajú zdrojové kódy programov použitých v práci. Pre viac informácií ohľadom obsahu elektronickej prílohy čitateľa odkazujeme na súbor „README.md“, ktorý sa nachádza v koreňovom adresári elektronickej prílohy.